Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Primož Lavrič

Spletno ogrodje za vizualizacijo volumetričnih podatkov z možnostjo oddaljenega sodelovanja

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Matija Marolt

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomskem delu se posvetite razvoju ogrodja, ki bo v brskalniku omogočalo vizualizacijo volumetričnih podatkov. Posvetite se predvsem posrednemu upodabljanju preko poligonskih modelov in v ogrodje vključite algoritem Marching cubes za pretvorbo v tovrstne modele. Testirajte hitrost algoritma v različnih implementacijah. Ogrodje naj omogoča tudi oddaljeno sodelovanje več uporabnikov, ki lahko delajo z istim modelom, v prostor postavljajo anotacije in sodelujejo v klepetu.

Najprej bi se zahvalil članom Laboratorija za računalniško grafiko in multimedije za podporo pri delu in družbo med pisanjem diplomske naloge. Zahvaljujem se mentorju doc. dr. Matiji Maroltu za podporo med pisanjem diplomskega dela. Še posebej pa bi se zahvalil as. dr. Cirilu Bohaku, ki je prispeval veliko idej in svojega časa k nastajanju diplomskega dela, hkrati pa zaupal v moje sposobnosti ter me usmerjal in s tem močno pripomogel h končni kakovosti dela.

Zahvali bi se Lani Beševič za pomoč in podporo pri pisanju ter strukturiranju dela in za popravljanje pravopisnih napak.

Na koncu bi se zahvalil še svoji družini, še posebej mami in očetu, ki sta mi omogočila študij ter me že od malega podpirata pri vseh zastavljenih ciljih, me spodbujata in verjameta v moje delo. Zahvaljujem se še svoji veliki sestri Anji, ki me je ogromno naučila, me spodbujala in podpirala, hkrati pa je v njeni družbi vedno zabavno.

Upam, da v zahvali nisem na nikogar pozabil. Če pa se mi je vseeno kdo izmuznil, hvala tudi tebi.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
2D	two-dimensional	dvodimenzionalen
3D	three-dimensional	tridimenzionalen
GPE	graphics processing unit	grafično procesna enota
MPUI	multi-level partition of unity implicits	algoritem implicitnih funkcij z večnivojsko particijo enote
RGBA	red green blue alpha	rdeča-zelena-modra prosojnost
CSS	cascading style sheets	

Povzetek

V delu predstavimo spletno vizualizacijsko ogrodje Med3D, razvito v sklopu te diplomske naloge, ki se primarno osredotoča na vizualizacije 3D mrežnih in volumetričnih medicinskih podatkov. Razvito ogrodje omogoča izris in interakcijo s tridimenzionalnimi mrežnimi podatki, posredno upodabljanje volumetričnih podatkov s pomočjo računsko učinkovitega algoritma Marching cubes in oddaljeno sodelovanje med uporabniki. Predstavimo implementacije različnih delov in funkcionalnosti ogrodja s poudarkom na realizaciji upodabljanja, uporabniškega vmesnika in oddaljenega sodelovanja. V sklopu upodabljanja opišemo implementirane funkcionalnosti razvitega upodobljevalnika, uporabljene pristope posrednega upodabljanja, neposredno upodabljanje in združevanja različnih tipov upodobitve s pomočjo odloženega upodabljanja. Podamo tudi performančno analizo mrežnega upodabljanja ter pretvorbo volumetričnih podatkov v mrežne, kjer testiramo različne implementacije (Java, C++, Javascript in ASM.js) algoritma Marching cubes. Predstavimo uporabniški vmesnik ogrodja s poudarkom na anotacijah, ki jih je mogoče pripeti na mrežni model. V sklopu oddaljenega sodelovanja opišemo realnočasovno delitev scene, klepet, omogočen znotraj ogrodja, in delitev anotacij med uporabniki. Podamo tudi performančno analizo oddaljenega sodelovanja, kjer smo analizirali odzivnost delitve scene.

Ključne besede: vizualizacija medicinskih podatkov, spletno vizualizacijsko ogrodje, Marching cubes, oddaljeno sodelovanje.

Abstract

In this paper, we present a web based visualization framework that was developed as part of this thesis and primarily focuses on the visualization of a medical 3D mesh and volumetric data. This framework offers visualization capabilities like 3D mesh rendering, scene navigation, indirect volume rendering via volume to mesh conversion, implemented with the computationally efficient Marching cubes algorithm, and online collaboration capability. We describe the implementations of different parts and functionalities of the framework, focusing on the implemented renderer, indirect volume rendering approaches, direct rendering and merging different types of rendering via deferred rendering. We present the performance evaluation of mesh rendering and four different implementations (Java, C++, Javascript and ASM.js) of volume to mesh conversion with the Marching cubes algorithm. We present the framework's user interface, focusing on annotations that can be pinned to the mesh. As part of remote rendering, we describe real-time scene sharing, implementation of chat and annotation sharing. We present the performance evaluation of remote collaboration where we tested the scene sharing responsiveness.

Keywords: visualization of medical data, web based visualization framework, Marching cubes, remote collaboration.

Poglavje 1

Uvod

Z razvojem novih tehnologij postaja vizualizacija tridimenzionalnih (3D) podatkov vse bolj nepogrešljiva in vse močnejše orodje za hitro obdelavo, preučevanje in razumevanje vizualiziranih podatkov. V vsakdanje življenje se vključuje na številnih področjih, kot so na primer gradbeništvo [1], meteorologija [2], geodezija [3], strojništvo, astronomija [4] in medicina [5, 6], saj omogoča prikaz in interakcijo s simuliranim okoljem, ki ga definirajo vhodni podatki. Simulacija omogoča nemoteno preučevanje podatkov v kontroliranem okolju, ki jih v realnosti morda ni mogoče preučevati zaradi fizikalnih, tehnoloških ali človeških omejitev oziroma prevelike potratnosti virov. Vizualizacije pogosto uporabimo za prikaz ogromnih količin podatkov, ki so v surovi obliki za človeka neinformativni, ko pa so abstrahirani in predstavljeni z vizualizacijo, postanejo veliko bolj informativni in razumljivi. Dober primer s področja medicine predstavljajo podatki, pridobljeni z različnimi metodami za 3D slikanje. Dobra interpretacija takšnih podatkov močno pripomore k določanju pravilne diagnoze pacientov in zmanjšuje potrebo po invazivnih posegih.

Pri 3D vizualizaciji se najpogosteje uporablja upodabljanje objektov, ki so predstavljeni z mrežo poligonov. Ti poligoni se ob upodobitvi projicirajo z željeno projekcijo (ponavadi perspektivna ali ortogonalna) in rasterizirajo. Ta pristop se je zaradi hitrosti upodabljanja, kompaktnosti podatkov, eno-

stavnega manipuliranja objektov in številnih ostalih dobrih lastnosti najbolj razširil. Ker je ta proces mogoče zelo dobro paralelizirati, so za ta namen razvili posebne grafične pospeševalnike, ki omogočajo hitro in vzporedno aritmetično računanje na več tisoč enotah.

Razvoj grafičnih pospeševalnikov je v zadnjih desetletjih zaznamoval napredek računalniške grafike. Povečevanje računske moči namreč omogoča vse hitrejši izris čedalje večjih količin podatkov z realnejšimi in bolj kompleksnimi metodami osvetljevanja, s katerimi lahko človeku na vse prijaznejši način predstavimo vizualizirane podatke. To je še posebej pomembno za simulacije, pri katerih želimo na čim boljši in razumljiv način iz različnih zornih kotov prikazati nek naraven pojav. Takšne simulacije se večinoma uporabljajo za poglobitev človekovega razumevanja in pripravo na problem v realnem svetu, zato je zelo pomembno, da na čim bolj iznajdljiv način natančno aproksimirajo realno stanje, hkrati pa izpostavijo podrobnosti, ki jih z opazovanjem v realnem svetu ne bi mogli opaziti. V medicini je še posebej pomembno, da ima zdravnik specialist na voljo čim bolj podroben vpogled v operiran del telesa, saj mu dober in razširjen pogled v pacientovo telo pomaga sprejemati boljše odločitve med samim posegom.

1.1 Cilji

Cilj te diplomske naloge je implementacija in razvoj spletnega vizualizacijskega ogrodja Med3D, katerega primarni namen je vizualizacija in interakcija z volumetričnimi medicinskimi podatki. Ti so zajeti s pomočjo različnih tehnik volumetričnega skeniranja.

Ogrodje temelji na predhodno razvitem ogrodju Neck Veins [7], ki je prav tako razvito za namene vizualizacije medicinskih podatkov, vendar gre v tem primeru za lokalno implementacijo v programskem jeziku Java. Zaradi želje po pokritosti širokega nabora platform in velikega števila različnih naprav je ogrodje Med3D implementirano na spletu in gnano v spletnih brskalnikih, ki so dandanes prisotni na skoraj vseh računalnikih in mobilnih napravah.

Hkrati spletna implementacija poveča doseg uporabnikov, saj omogoča enostavno dostopanje do ogrodja in začetek uporabe brez potrebe po predhodni namestitvi.

Poleg vizualizacije želimo v ogrodju podpreti tudi možnost lokalne obdelave podatkov v okviru zmožnosti naprave, na kateri deluje. Velik delež medicinskih podatkov namreč ni podan v mrežni obliki, ki jo bo implementirano ogrodje primarno izrisovalo. Takšne podatke je pred vizualizacijo treba dodatno obdelati in pretvoriti v mrežno obliko. Z lokalnim obdelovanjem podatkov se lahko izognemo preveliki komunikaciji s strežnikom, ki bi močno obremenila tako uporabnikovo napravo kot tudi strežnik. V obsegu diplomskega dela smo v ogrodju implementirali osnovno pretvorbo iz volumetričnih podatkov, podanih s 3D skalarnim poljem, v mrežni model s pomočjo algoritma Marching cubes.

V ogrodju želimo podpreti tudi oddaljeno sodelovanje med uporabniki. Oddaljeno sodelovanje omogoča delitev pogleda in realnočasovno sinhroniziranje podatkov med povezanimi uporabniki. Takšno oddaljeno sodelovanje uporabnikom omogoča skupno in posledično hitrejšo interpretacijo podatkov, hkrati pa olajša dostop do znanja in mnenja oddaljenih specialistov. Sodelovanje z delitvijo pogleda pogosto ni dovolj za hitro in učinkovito interpretacijo podatkov. Iz tega razloga smo v ogrodju omogočili deljenje in dodajanje tekstovnih anotacij na mrežni model ter klepetalnico, preko katere si bodo povezani uporabniki lahko delili tekstovna sporočila.

Vse pomembnejše funkcionalnosti ogrodja, kot so upodabljanje, pretvorba mrežnih podatkov in oddaljeno sodelovanje, smo performančno analizirali in ovrednotili. Tako lahko razvito ogrodje lažje primerjamo z že razvitimi rešitvami, hkrati pa nam je analiza pomagala izpostaviti omejitve, ki jih prinaša spletna implementacija.

Ključno motivacijo za razvoj ogrodja predstavlja želja po izboljšavi trenutno uporabljenih pristopov ter s tem zdravnikom olajšati delo, jih zbližati in jim omogočiti možnost nudenja kakovostnejše zdravstvene pomoči.

1.2 Struktura dela

Delo je razčlenjeno na pet poglavij. Uvodu sledi drugo poglavje, ki predstavlja pregled področja in vizualizacijo medicinskih podatkov, različne pristope posrednega in neposrednega upodabljanja ter različne spletne tehnologije. V tretjem poglavju predstavimo lokalni del razvitega ogrodja, ki obsega upodabljanje in uporabniški vmesnik. Podamo tudi performančno analizo upodabljanja. V četrtem poglavju opišemo oddaljeno sodelovanje, omogočeno znotraj ogrodja. Podrobneje predstavimo delitev scene, klepet in delitev anotacij ter podamo performančno analizo komunikacije. V zaključku povzamemo rezultate dela in podamo ideje za nadaljnji razvoj.

Poglavje 2

Pregled področja

Koncept vizualizacije obstaja že več stoletij in tisočletij ter obsega vse od kartografije, s pomočjo katere so že stari Grki konstruirali zemljevide [8], do tehničnih načrtov in upodobitev s pomočjo računalniške tehnologije. V tem delu se bomo osredotočili na slednje. Dandanes se večina vizualizacij ustvari s pomočjo računalniške grafike. Ta omogoča hitro, dinamično in kakovostno vizualiziranje ogromnih količin podatkov. To je zelo pomembno, saj zaradi narave človeka veliko lažje razumemo in interpretiramo abstrahirane vizualne upodobitve podatkov kot podatke v surovi obliki.

2.1 Splošen uvod v vizualizacijo

Področje računalniškega vizualiziranja podatkov ima že dolgo zgodovino. Večji premiki na tem področju so se začeli sredi sedemdesetih let dvajsetega stoletja, ko se je začela vizualizacija uporabljati v znanstvenih in inženirskih panogah za računalniško modeliranje in simuliranje [9]. Kasneje je s pospešenim razvojem računalniške tehnologije in s čedalje večjimi zbirkami podatkov potreba po dobrih vizualizacijah le še narastla. Hkrati je razvoj računalniške tehnologije omogočil kompleksnejše in natančnejše modele vizualiziranja. Kot smo že omenili, se je v 3D vizualizaciji najbolje uveljavila tehnika upodabljanja scene s 3D objekti, sestavljenimi iz poligonskih mrež,

lučmi, ki nosijo informacije o osvetlitvi, in kamero, ki predstavlja točko in smer pogleda na vizualizirano sceno. Preden se lahko scena izriše na platno, jo je treba ustrezno preslikati na dvodimenzionalno (2D) ravnino. To dosežemo s pomočjo algoritmov za preiskovanje prostora in namenskih programov, senčilnikov, za grafično procesno enoto (GPE). Prvi so zadolženi za čim hitrejše določanje podmnožice podatkov, ki so vidni iz trenutnega pogleda, kar nam lahko bistveno zmanjša računsko in prostorsko kompleksnost upodobitve, drugi pa za izračun končne upodobitve podmnožice podatkov na 2D ravnini s pomočjo algoritmov za osvetljevanje in senčenje. Uporaba naprednejših algoritmov nam omogoča bolj kakovostno in natančno upodobitev scene. Vendar pa gre kakovost upodobitve z roko v roki s hitrostjo izrisa. Kakovostnejša upodobitev je ponavadi zaradi natančnosti veliko bolj časovno potratna, zato je treba pri upodabljanju paziti, da so uporabljeni algoritmi dovolj hitri za namen vizualizacije. Pri realnočasovnih simulacijah namreč želimo, da lahko sceno upodobimo v zelo kratkem času (nekaj deset milisekund), torej vsaj tridesetkrat na sekundo, saj želimo, da opazovalec dobi vtis, da se objekti simulacije zvezno premikajo. Na področjih, kjer je pomembna zgolj kakovost končnega izrisa, pa si lahko privoščimo tudi nekaj urni izris z uporabo zelo kompleksnih in natančnih algoritmov.

Poleg omenjene tehnike se za volumetrične podatke pogosto uporablja tudi tehnika neposrednega upodabljanja z metanjem žarkov [10]. Algoritmi, ki temeljijo na omenjeni tehniki generirajo vizualizacijo s pomočjo navideznih žarkov, ki so iz izhodišča kamere poslani v prostor. Med potovanjem skozi prostor vsak žarek z vzorčenjem prostora izračuna svoj prispevek, ki se na končni sliki odraža kot barva pripadajočega slikovnega elementa – piksla.

2.2 Vizualizacija medicinskih podatkov

Z razvojem računalniške grafike postajajo medicinske vizualizacije podatkov vse močnejše orodje zdravnikov specialistov pri postavljanju hitre in točne diagnoze. Prav tako pa zmanjšujejo potrebo po invazivnejših posegih in poma-

gajo pri načrtovanju in pripravi na morebiten kirurški poseg, saj omogočajo natančnejši vpogled v notranjost človeškega telesa.

Velik delež medicinskih podatkov predstavljajo 3D skalarna polja, ki so zajeta s pomočjo različnih tehnik volumetričnega skeniranja, kot na primer tehnike računske tomografije (angl. computed tomography – CT) [11, 12], pozitronska emisijska tomografija (angl. positron emission tomography – PET) [13], slikanje s pomočjo magnetne resonance (angl. magnetic resonance imaging – MRI) [14] in ultrazvok (angl. ultrasound – US) [15]. Tako zajeti podatki so ponavadi precej obsežni, tudi do nekaj 100 MB ali celo nekaj GB, in posledično njihova vizualizacija predstavlja svojevrsten tehnični izziv. V ta namen se vizualiziranje in interpretiranje tako zajetih podatkov izvaja na specializirani, dovolj zmogljivi strojni in programski opremi. Takšna oprema je draga in stacionarna ter zdravnikom ne omogoča podajanja diagnoze in priprave na poseg na daljavo, kar otežuje pridobivanje drugega mnenja ali mnenja specialista na oddaljeni lokaciji. Ta problematika se je v zadnjih letih začela naslavljati, kar nakazujejo številna ogrodja za medicinske vizualizacije, kot na primer SimVascular [16] in Exposure Renderer [17], ki omogočajo vizualiziranje na nekoliko zmogljivejših osebnih računalnikih. Vsa omenjena ogrodja volumetrične podatke vizualizirajo z različnimi tehnikami, ki spadajo med pristope posrednega ali neposrednega upodabljanja.

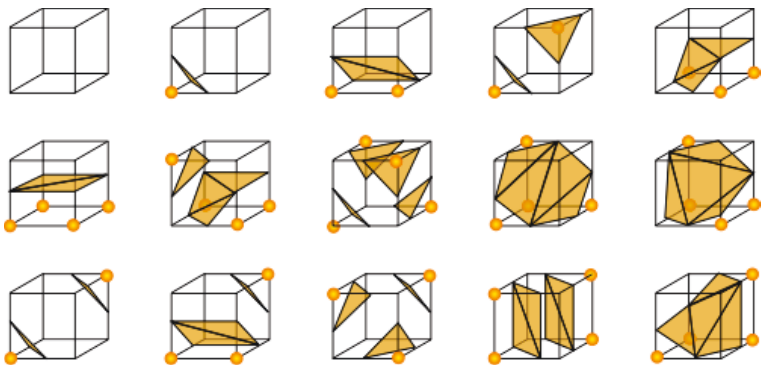
2.3 Pristopi posrednega upodabljanja

Za poenostavitev upodabljanja in zmanjševanje prostorske kompleksnosti volumetričnih podatkov se pogosto zatečemo k pristopom posrednega upodabljanja. Pri takšnih pristopih skalarno polje najprej pretvorimo v predstavitev s 3D primitivi, ponavadi trikotniki, ki jih vpnemo v mrežo okoli aktivnih vokslov – 3D pikslov skalarnega polja, določenih s postavljenim pragom intenzitete. Posledično močno zmanjšamo prostorsko kompleksnost, saj je predstavitev s 3D primitivi prostorsko veliko bolj varčna kot predstavitev s skalarnim poljem. Hkrati je upodabljanje 3D primitivov na današnji strojni

opremi veliko hitrejše od pristopov z neposrednim upodabljanjem.

2.3.1 Metoda Marching cubes

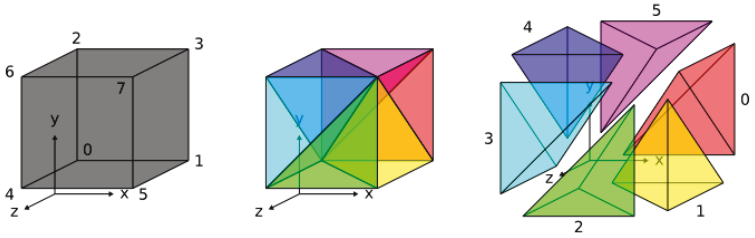
Najpogosteje uporabljena metoda za pretvorbo volumetričnih podatkov je metoda Marching cubes [18]. Kot namiguje že ime, se omenjena metoda sprehodi skozi 3D skalarno polje z navidezno kocko. Ob vsakem koraku se kocka poravna tako, da njena oglišča sovpadajo z osmimi sosednjimi voksli. Ti voksli lahko zavzemajo enega od dveh stanj, lahko so bodisi aktivni bodisi neaktivni. Njihovo aktivnost določa prag intenzitete, ki ga uporabnik določi glede na željeno vizualizacijo. Navidezna kocka ima torej lahko 2^8 različnih konfiguracij aktivnih vokslov. Za vsako izmed konfiguracij imamo predefinirano enolično preslikavo v skupek poligonov, ki se pravilno ovije okoli aktivnih vokslov (glej sliko 2.1). Ujemajoči skupek poligonov je nato treba le še ustrezno zamakniti glede na trenuten položaj navidezne kocke v skalarnem polju. Tako generirane skupke med iteriranjem združujemo in na koncu dobimo poligonsko mrežo, ki je vpeta okoli vseh aktivnih vokslov.



Slika 2.1: Primer 15 konfiguracij z označenimi aktivnimi voksli. (`https://en.wikipedia.org/wiki/Marching_cubes`)

2.3.2 Metoda Marching tetrahedra

Z iztekom patenta nad algoritmom Marching cubes se je, izhajajoč iz metode Marching cubes, začel uporabljati algoritem Marching tetrahedra [19]. Pri algoritmu Marching tetrahedra navidezno kocko dodatno razčlenimo na šest nepravilnih tetraedrov, ki jih dobimo, če navidezno kocko trikrat prerežemo po eni diagonali para nasprotnih ploskev in po eni izmed glavnih diagonal kocke (glej sliko 2.2). Tako si vsi tetraedri delijo eno glavno diagonalo in dve oglišči navidezne kocke. S takšno razdelitvijo smo poleg originalnih dvanajstih stranic kocke pridobili še dodatnih sedem stranic, kar nam omogoča, da natančneje definiramo konfiguracije kocke s pomočjo šestih tetraedrov, izmed katerih lahko vsak zavzema eno izmed šestnajstih konfiguracij. V primerjavi z Marching cubes nam to prinese natančnejše vzorčenje skalarnega polja in posledično natančneje vpeto mrežo primitivov. Vendar pa ima omenjena metoda tudi slabosti. Zaradi razdelitve na tetraedre moramo v vsaki iteraciji pridobiti šest konfiguracij namesto ene, kar poveča časovno kompleksnost algoritma. Hkrati pa se lahko pri ostrejših prehodih površine pojavijo nazobčane strukture.



Slika 2.2: Razdelitev navidezne kocke na 6 nepravilnih tetraedrov. (`https://dune-project.org`)

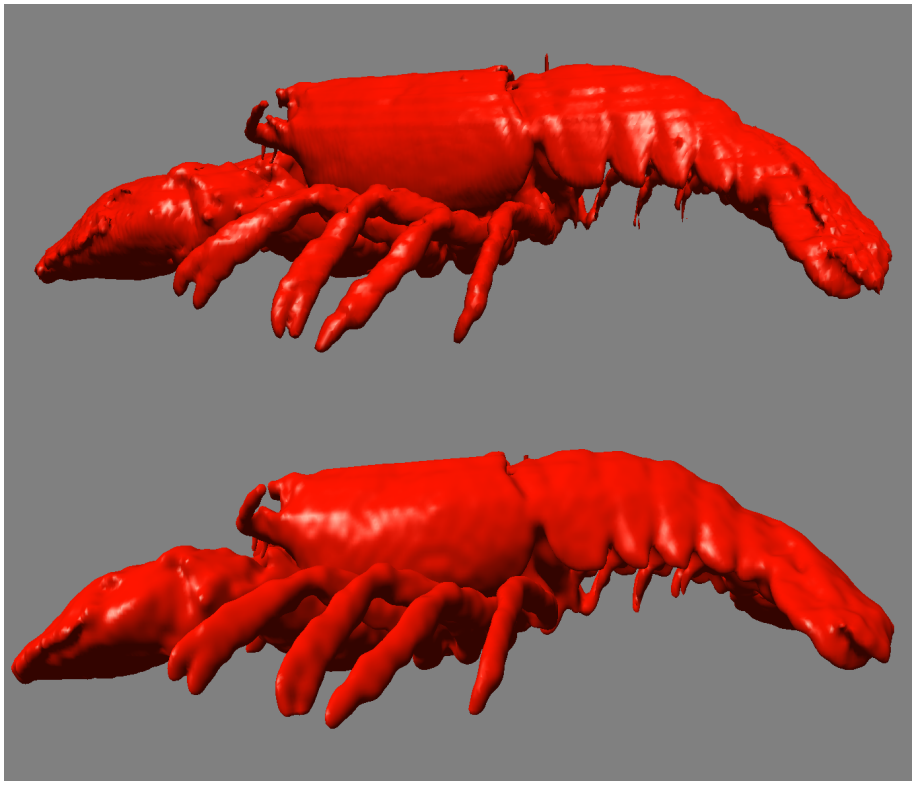
2.3.3 Metoda MPUI

Poleg omenjenih algoritmov se med pristopi posrednega upodabljanja uveljavlja tudi algoritem implicitnih funkcij z večnivojsko particijo enote (angl.

multi-level partition of unity implicits – MPUI) [20], prvič predstavljen leta 2003. Ta metoda odpravlja glavno pomanjkljivost predhodno omenjenih metod, katerih težava je nezmožnost modeliranja ostrih značilnosti predmeta (glej sliko 2.3). Algoritem definirajo trije ključni elementi: lokalne implicitne funkcije, ki modelirajo lokalno obliko objekta, utežne funkcije, namenjene povezovanju in združevanju lokalnih funkcij, ter osmiško drevo, ki omogoča preiskovanje in adaptiranje algoritma glede na kompleksnost lokalne oblike. Vhodni podatki algoritma morajo biti podani v obliki oblaka točk, kjer vsaka točka vsebuje tako pozicijo kot normalo na površino objekta. To pomeni, da moramo skalarno polje predhodno dodatno obdelati in pretvoriti v obliko, primerno za algoritem. Zaradi obsežne priprave podatkov in kompleksnega modeliranja je metoda računsko in prostorsko zelo zahtevna, zato se nanjo zanašamo le, ko imamo dovolj virov in časa za procesiranje podatkov. Hkrati je pretvorbo treba ponoviti vsakič, ko spremenimo množico aktivnih vokslov, zato se v primeru pogostih pretvorb raje poslužujemo pristopov neposrednega upodabljanja.

2.4 Pristopi neposrednega upodabljanja

Pristopi posrednega upodabljanja in predstavitev volumetričnih podatkov v obliki mreže primitivov so velikokrat nepogrešljivi zaradi številnih dobrih lastnosti, vendar pa imajo takšni pristopi tudi slabosti. Kakovost modela je namreč pogosto pogojena s številom generiranih primitivov, ki lahko pri večjih skalarnih poljih hitro preseže zgornjo mejo. Hkrati pa z upodabljanjem primitivov, ki predstavljajo zgolj površino objekta, izgubimo veliko pomembnih informacij, vsebovanih v podatkih. Nasprotje pristopov s posrednim upodabljanjem predstavljajo pristopi z neposrednim upodabljanjem [22, 23, 24], ki volumetričnih podatkov pred upodabljanjem ne pretvarjajo v drugo obliko, temveč z različnimi pristopi upodabljanja vizualizirajo kar surovo obliko skalarnega polja. Pri takšnem upodabljanju potrebujemo neke vrste prenosno funkcijo za preslikavo vrednosti skalarnega polja v barvo in



Slika 2.3: Primerjava rezultatov poligonizacije z algoritmom Marching cubes (zgoraj) in algoritmom MPUI (spodaj) [21].

prosojnost (RGBA). Ob upodobitvi se pridobljene vrednosti odražajo na pikslih generirane slike, kot definira tehnika upodabljanja.

2.4.1 Upodabljanje s pomočjo tekstur

Upodabljanje s pomočjo tekstur [25] velja za najpreprostejšo tehniko neposrednega upodabljanja in posledično za eno izmed najhitrejših, saj današnji računalniki omogočajo strojno podprto upodabljanje tridimenzionalnih tekstur z uporabo grafičnih pospeševalnikov. Če 3D teksture niso podprte, jih je mogoče nadomestiti s serijo ustrezno razvrščenih 2D tekstur, saj gre v obeh primerih za upodabljanje skupka rezin v 3D prostoru. Obstajata dve metodi upodabljanja s pomočjo tekstur. Pri prvi so rezine ves čas usmerjene proti

kameri, tekstura pa se rotira relativno na pogled kamere [26]. Pri drugi so rezine poravnane s stacionarnim objektom v prostoru, ki predstavlja upodabljane volumetrične podatke [26]. Prva metoda omogoča zgolj uporabo 3D teksture, druga pa podpira tudi serijo 2D tekstur. Ta tehnika je hitra, vendar pa ima tudi slabosti, kot so na primer velika prostorska zahtevnost zaradi nezmožnosti uporabe učinkovitih podatkovnih struktur, statična gostota vzorčenja in omejena uporaba filtrov.

2.4.2 Tehnika Splatting

Po hitrosti primerljiva je tehnika Splatting [27, 28], predstavljena leta 1990. Osnovna ideja te tehnike je, da vsak voksel "vržemo" v platno in na platnu upoštevamo njegov prispevek. To realiziramo tako, da skalarno polje pretvorimo v serijo prekrivajočih se baznih funkcij. Najpogosteje se uporabi Gaussovo jedro, katerega amplitude izrazimo z vrednostmi vokslov. Tako generirane bazne funkcije najprej razvrstimo glede na oddaljenost od platna, nato pa jih projiciramo na platno in združujemo njihov prispevek. Omenjeni prispevki se tako na končni sliki odražajo v obliki barv pikslov. Obstajajo tudi številne izboljšave osnovne tehnike. Eno izmed njih predstavlja generiranje senc [29], ki pripomorejo k pristnosti upodobitve. Sence realiziramo tako, da za vsakega izmed pikslov hranimo faktor pojemanja svetlobe, ki ga definirajo predhodno upodobljene bazne funkcije in se odraža na vseh sledečih funkcijah.

2.4.3 Tehnika Shear-Warp

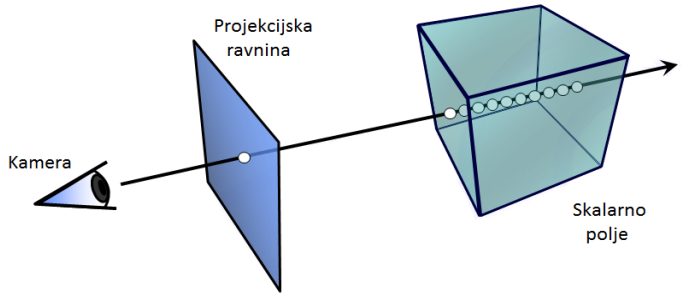
Za generiranje različnih pogledov na objekt, podan z volumetričnimi podatki, moramo voksle pred projiciranjem na 2D ravnino najprej rotirati, kar nam oteži proces projiciranja. Tehnika Shear-Warp [30], ki sta jo leta 1994 predstavila Lacroute in Levoy, rešuje ta problem, tako da nadomesti rotacijo volumna z zamikanjem njegovih rezin (angl. shearing), ki po transformaciji ostanejo poravnane z dvema koodrinatnima osema. S tako zamaknjenimi

rezninami se lahko hitro ustvari začasna vmesna slika, ki pa je popačena. Ta popačenost se naslovi v zadnjem koraku, imenovanem ukrivljanje (angl. warping), ko se vmesna slika transformira v končno sliko pogleda. Opisana tehnika je še dandanes prepoznana kot najhitrejša za upodabljanje volumetričnih podatkov. Vendar pa ima tudi nekaj slabosti. Upodabljanje vključuje dva koraka vzorčenja, kar lahko povzroči izgubo podrobnosti in zameglitev končnega izrisa. Zaradi načina upodobitve osnovna implementacija ne podpira trilinearne interpolacije, ampak zgolj bilinearno na nivoju posamezne reznine. Med ortogonalnimi rezinami se uporablja filter najbližji sosed (angl. nearest neighbor). To povzroča napake pri izrisu podatkov, kjer imamo strme prehode barve oziroma prosojnosti.

2.4.4 Tehnika metanja žarkov

Za tehniko z najbolj kakovostnimi rezultati velja tehnika metanja žarkov (angl. Ray casting). V kontrastu z ostalimi metodami, ki upodobijo pogled s pomočjo projiciranja na platno, ta metoda pogled upodobi z "metanjem" navideznih žarkov iz izhodišča kamere. Žarki na poti skozi prostor z adaptivnim oziroma fiksnim intervalom vzorčijo prostor in akumulirajo prispevek obiskanih vokslov skalarnega polja, ki se odraža na končni barvi in prosojnosti žarka (glej sliko 2.4). Pri prispevku lahko upoštevamo tudi material objekta in vpliv luči, kar še dodatno poveča računsko kompleksnost algoritma. Najpogosteje vsakemu pikslu končne upodobitve pripada vsaj en žarek, ki mu ob koncu upodobitve priskrbi omenjeno barvno in prosojnostno informacijo. Natančnost končne upodobitve je torej pogojena s številom pikslov in posledično s številom žarkov, kar tudi definira računsko kompleksnost algoritma. Kljub visoki računski kompleksnosti pa neodvisnost žarkom omogoča visoko stopnjo paralelizacije, kar pridoma izkoriščamo z grafičnimi pospeševalniki. Poleg paralelizacije lahko pohitritev dosežemo tudi z drugimi optimizacijskimi metodami, kot je na primer zgodnje prekinjanje žarkov, ko ti zapustijo območje volumetričnega objekta, in dobro predstavitvijo podatkov z osmiškim drevesom vokslov [31], ki nam poleg prostorsko učinkovitega zapisa omogoča

preskakovanje praznega prostora, kar lahko bistveno pohitri izvajanje algoritma. Tehnika metanja žarkov in ostale tehnike posrednega upodabljanja postajajo vse dostopnejše tudi na spletu, saj z razvojem tehnologije brskalniki omogočajo čedalje več funkcionalnosti upodabljanja na spletu, potrebnih za realizacijo metod, ki jih definirajo tehnike.



Slika 2.4: Vzorčenje skalarnega polja s pomočjo metanja žarka. (`http://www.cb.uu.se/~aht/Vis2014/SeSE_RenderingTechs_20141112.pdf`)

2.5 Spletne tehnologije

Generirane vizualizacije želimo pogosto deliti in interpreterati z ostalimi, vendar pa to zaradi geografske oddaljenosti ni vedno mogoče. S pomočjo svetovnega spleta, ki se je v zadnjih desetletjih zelo razširil in je dandanes na voljo že za več kot 40 % populacije, lahko naslovimo ta problem s pomočjo oddaljene komunikacije in sodelovanja med uporabniki. Hkrati pa je poganjanje vizualizacijskih ogrodij in aplikacij nasploh na spletu vse bolj podprto s strani spletnih brskalnikov. V namen poenostavitve implementacije spletnih aplikacij so bila razvita številna ogrodja in knjižnice, ki poenostavijo razvoj uporabniškega vmesnika in omogočajo uporabo grafičnega pospeševalnika v brskalniku. Za nas najpomembnejšo knjižnico predstavlja knjižnica WebGL (angl. Web Graphics Library) [32], ki nam omogoča upodabljanje v brskalniku.

2.5.1 Knjižnica WebGL

Knjižnica WebGL, ki je v brskalnikih na voljo od marca 2011, predstavlja vmesnik za namensko programiranje grafičnega pospeševalnika v programskem jeziku Javascript. Knjižnica nam omogoča 2D in 3D upodabljanje v vseh sodobnih brskalnikih brez uporabe namenskih vtičnikov. Aplikacija, ki uporablja WebGL, sestoji iz dveh ključnih elementov. Prvi je kontrolni del, ki je spisan v programskem jeziku Javascript in skrbi za prenašanje podatkov med grafičnim pospeševalnikom in računalnikom ter konfiguriranje nastavitev, ki vplivajo na generirano upodobitev. Drugi element pa predstavljajo senčilni programi, spisani v posebnem jeziku (angl. Graphics Library Shading Language - GLSL) [33], definiranem s standardom OpenGL, ki se v veliki meri zgleduje po programskem jeziku C. Takšni senčilni programi se ob začetku upodabljanja prevedejo v obliko, primerno za uporabljani grafični pospeševalnik, saj jih med upodabljanjem ta uporabi za generiranje želenega izrisa. Knjižnica WebGL je zaradi narave računalniškega upodabljanja precej kompleksna in nizkonivojska. Zaradi tega so se razvila ogrodja, kot so Three.js [34] in BabylonJS [35], ki abstrahirajo uporabo WebGL in močno olajšajo uporabo različnih funkcionalnosti upodabljanja. Hkrati nam ponujajo hierarhično organizacijo scene in veliko pomožnih metod za manipulacijo objektov v 3D prostoru. Kljub temu smo se pri implementaciji upodobljevalnika v našem ogrodju raje zanesli na neposredno uporabo knjižnice WebGL, saj nam to omogoča, da upodobljevalnik bolje prilagodimo našim zahtevam in posledično pohitrimo upodabljanje. Hkrati se tako znebimo obsežnega dela neuporabljene kode, ki bi upočasnjevala nalaganje ogrodja, saj bi uporabili zgolj manjšo podmnožico funkcionalnosti, ki jih ponujajo omenjena ogrodja. Tako realiziran upodobljevalnik smo v ogrodje integrirali s pomočjo ogrodja AngularJS [36].

2.5.2 Ogrodje AngularJS

Ko razvijamo aplikacijo s pomočjo knjižnice WebGL, želimo pogosto imeti v aplikaciji zgolj eno platno. To pomeni, da je treba aplikacijo zasnovati in razviti kot aplikacijo, ki se izvaja znotraj ene strani (angl. single page application – SPA). To lahko predstavlja precejšen tehnični izziv. Proces razvoja obsežnih aplikacij, kot so ogrodja za vizualizacijo, je namreč brez uporabe dobrih praks pri načrtovanju in razvoju zelo počasen, nepredvidljiv in nagnjen k napakam. Zato se pogosto zanašamo na uveljavljena ogrodja, kot je AngularJS, ki nam ob razvoju vsiljujejo uporabo dobrih in preverjenih praks. AngularJS je odprtokodno ogrodje, podprto s strani Googla, ki nam v aplikaciji doda dodaten nivo abstrakcije v obliki gnezdenih modulov, kot so direktive (angl. directive), kontrolerji (angl. controller) in storitve (angl. service), hkrati pa nam omogoča enosmerno ali dvosmerno podatkovno povezovanje (angl. data binding), ki namesto nas sinhronizira podatke, deljene med različnimi moduli aplikacije. Ogrodja AngularJS smo se poslužili tudi pri razvoju ogrodja Med3D. Dober primer učinkovite uporabe AngularJS predstavljajo grafični navigatorji scene, ki so implementirani povsem neodvisno od ogrodja in so z njim povezani zgolj s podatkovnim povezovanjem, preko katerega mu zagotavljajo informacije o svojem stanju. Ogrodje AngularJS pogosto uporabimo v kombinaciji z ogrodjem Bootstrap [37], saj nam ta priskrbi številne segmente uporabniškega vmesnika, ki jih lahko s pomočjo ogrodja AngularJS enostavno povežemo v kakovosten uporabniški vmesnik.

2.5.3 Ogrodje Bootstrap

Pomemben del vizualizacijskih ogrodij predstavlja uporabniški vmesnik. Ta, kadar je pravilno in konsistentno zasnovan, uporabnikom omogoča hitro in učinkovito manipulacijo vizualizacij in podatkov. Zasnova takšnega uporabniškega vmesnika je pogosto dolgotrajna in zahtevna. Zato se pri razvoju spletnih ogrodij pogosto obrnemo na odprtokodno ogrodje Bootstrap, ki vsebuje številne med seboj konsistentne predloge segmentov uporabniškega

vmesnika. Te je zaradi lastnosti jezika CSS (angl. cascading style sheets) [38] mogoče enostavno prilagajati in nadgrajevati, hkrati pa lahko za njimi vpeljemo dodatno logiko in dinamičnost s pomočjo jezika Javascript. V ogrodju Med3D smo tako uporabili številne predloge, na primer spustne menije, gumbe, navigacijsko vrstico in veliko ostalih, ki smo jih grafično in funkcionalno prilagodili, tako da ustrezajo stilu in potrebam ogrodja. Uporabniški vmesnik in stanje ogrodja nasploh želimo pogosto sinhronizirati s podatki ostalih uporabnikov ogrodja. To dosežemo s pomočjo strežnika, ki je v našem ogrodju realiziran s platformo Node.js [39].

2.5.4 Platforma Node.js

Dandanes je skoraj vsaka spletna aplikacija dodatno podprta s strani strežnika. Strežnik omogoča dolgotrajno shranjevanje podatkov, interakcijo med uporabniki in mnogo ostalih storitev, ki jih ni mogoče realizirati izključno na strani uporabnika v spletnem brskalniku. Spletni brskalnik namreč ponuja dostop zgolj do omejenih virov, ki so pogojeni z uporabniško strojno opremo, omenjeni viri pa pogosto ne zadostujejo za izvajanje kompleksnejših operacij. Implementacija strežnika še zdaleč ni trivialna naloga, zato se zanesemo na ogrodja, ki omogočajo hiter in strukturiran razvoj strežniške strani aplikacije. V zadnjih letih se v ta namen večinoma uporablja odprtokodno ogrodje Node.js [39], ki za rokovanje z zahtevki uporablja asinhron in dogodkovno gnan model, ki ne blokira izhodno-vhodnih naprav, kar omogoča zelo visoko skalabilnost. Hkrati ponuja široko zbirko zanesljivih odprtokodnih modulov, ki jih lahko uporabimo v aplikaciji za poenostavitev in pohitritev implementacije strežniške strani aplikacije. V ogrodju Med3D je strežniška podpora s pomočjo Node.js ključnega pomena za realiziranje oddaljenga sodelovanja, saj omogoča hitro in učinkovito posredovanje podatkov med uporabniki, kar je ključnega pomena pri realnočasovni delitvi vizualizacije.

V sklopu pregleda področja smo predstavili uvod v vizualizacijo medicinskih podatkov, v katerem smo podali razlog za razvoj ogrodja Med3D. Predstavili smo tudi vse glavne metode, pristope in tehnologije, ki smo jih

preučili in si z njimi pomagali pri razvoju vizualizacijskega ogrodja Med3D. Podrobneje smo se oprli na različne upodobitvene pristope, ki so nam služili kot osnova za dobro vizualiziranje medicinskih podatkov, in spletne tehnologije, ki smo jih uporabili za implementacijo ogrodja na spletu. S pomočjo pregleda področja smo si zastavili cilje, ki smo jih želeli doseči, hkrati pa nam le-ta služi kot dobra podlaga za vse nadaljnje delo.

Poglavje 3

Vizualizacijsko ogrodje na spletu

V tem poglavju bomo predstavili spletno vizualizacijsko ogrodje Med3D in omenili nekaj že razvitih ogrodij, ki bodo služila kot referenčna točka pri evalvaciji ogrodja Med3D. Najprej bomo podali razloge, zakaj je to ogrodje potrebno, kako bo uporabljeno, kakšne so njegove glavne funkcionalnosti, zakaj smo se odločili za implementacijo na spletu in kakšne prednosti oziroma slabosti to prinaša. Spustili se bomo tudi v arhitekturo ogrodja, kjer se bomo navezali na module, ki ga sestavljajo, podrobneje predstavili njihove funkcionalnosti in njihovo medsebojno interakcijo.

V nadaljevanju se bomo poglobili v samo jedro ogrodja, ki ga predstavlja upodabljanje 3D scene. Pri upodabljanju bomo najprej predstavili osnovne koncepte 3D računalniške grafike, kot so kamera, organizacija scene, predstavitev objektov, prosojnost ter metode osvetljevanja in senčenja. Tako bomo postavili temelje za razumevanje posrednega in neposrednega upodabljanja 3D volumetričnih podatkov. Pri posrednem upodabljanju se bomo podrobneje poglobili v algoritem Marching cubes in njegovo paralelizacijo. Na kratko bomo predstavili tudi algoritem MPUI, ki velja za eno izmed naprednejših metod pretvorbe volumetričnih podatkov v mrežni model. V poglavju neposrednega upodabljanja pa se bomo navezali na tehniko Splat-

ting in tehniko metanja žarkov (angl. Ray casting), s katerima nameravamo v prihodnosti razširiti naše vizualizacijsko ogrodje, ki je za njihovo implementacijo že pripravljeno.

Predstavili bomo tudi uporabniški vmesnik, ki je ključni del ogrodja, saj močno pripomore k dobri uporabniški izkušnji. Podrobneje bomo predstavili podporo uporabniškim anotacijam z možnostjo pripenjanja na izrisano 3D mrežo. To namreč uporabniku omogoča, da lažje interpretira velike vizualizacije in označuje interesne točke, ki jih lahko nato deli z ostalimi uporabniki ogrodja.

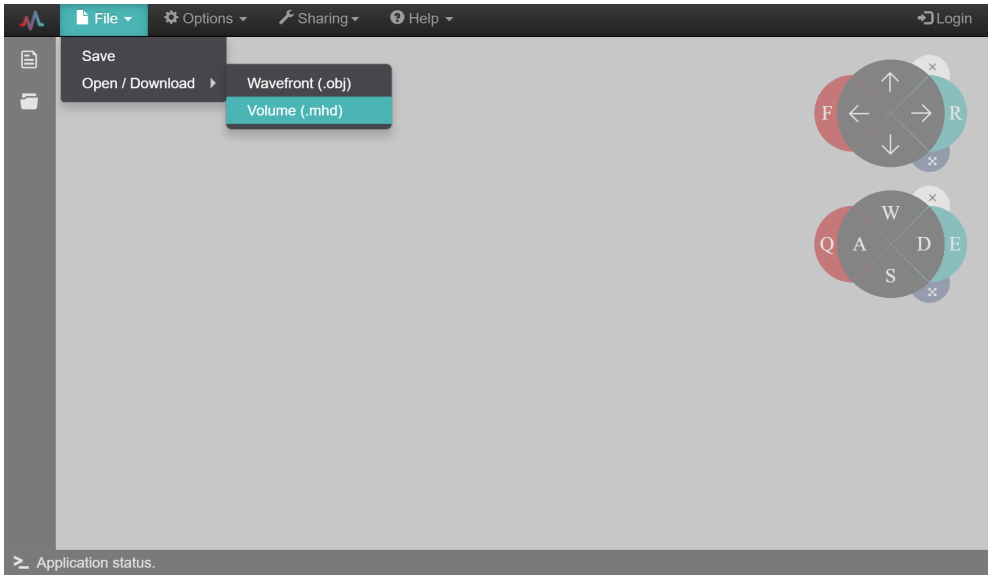
Na koncu poglavja pa sledi performančna analiza sistema, kjer bomo predstavili rezultate performančnih testov zahtevnejših funkcionalnosti ogrodja. Tu bomo podrobneje preučili spletni implementaciji algoritma Marching cubes s pomočjo jezika Javscript in ASM.js ter ju primerjali z lokalnima implementacijama v jeziku Java in C++.

3.1 Predstavitev ogrodja

Za namene vizualizacije volumetričnih podatkov je bilo razvitih več ogrodij in aplikacij. Exposure Renderer [17] in SimVascular [16] sta namenjeni lokalni vizualizaciji volumetričnih podatkov in podpirata napredne osvetlitvene tehnike. ParaView [40] omogoča paralelno obravnavo in vizualiziranje obsežnih podatkov z uporabo namenskih strežnikov. Predstavljena orodja ne podpirajo širokega nabora platform in so omejena na uporabo na namiznih računalnikih ali prenosnikih. Ogrodje ParaViewWeb [41] omogoča uporabo preko spletnega brskalnika s pomočjo oddaljenega upodabljanja, a je prav zato njena uporaba nekoliko omejena zaradi manjše odzivnosti. Predstavljeno orodje sicer deluje na veliko platformah, a enako kot prejšnja ne omogoča oddaljenega sodelovanja med uporabniki.

V kontrastu z že razvitimi ogrodji smo ogrodje Med3D (glej sliko 3.1), ki delno temelji na predhodno razvitem ogrodju Neck Veins, razvili z namenom vizualizacije medicinskih volumetričnih podatkov neposredno v spletnem br-

skalniku z možnostjo oddaljenega sodelovanja med uporabniki. Spletna implementacija poveča doseg uporabnikov in omogoča uporabo tudi na mobilnih napravah. Uporabnik lahko s pomočjo ogrodja lokalno obdeluje in vizualizira podatke v okviru zmožnosti naprave, na kateri deluje. To omogoča visoko stopnjo interaktivnosti vizualizacije, kar posledično vodi k boljši uporabniški izkušnji. Oddaljeno sodelovanje, ki ga ponuja ogrodje, omogoča deljenje pogleda, anotacij in klepet z oddaljenimi uporabniki. Tako oddaljenim uporabnikom omogočimo dober vpogled v dejansko stanje pacienta, gostujočemu uporabniku pa omogočimo pridobitev bolj kakovostnega in zanesljivega mnenja oddaljenega zdravnika specialista. Prenos informacij in znanja dodatno pospeši klepet, vgrajen v vizualizacijsko ogrodje.



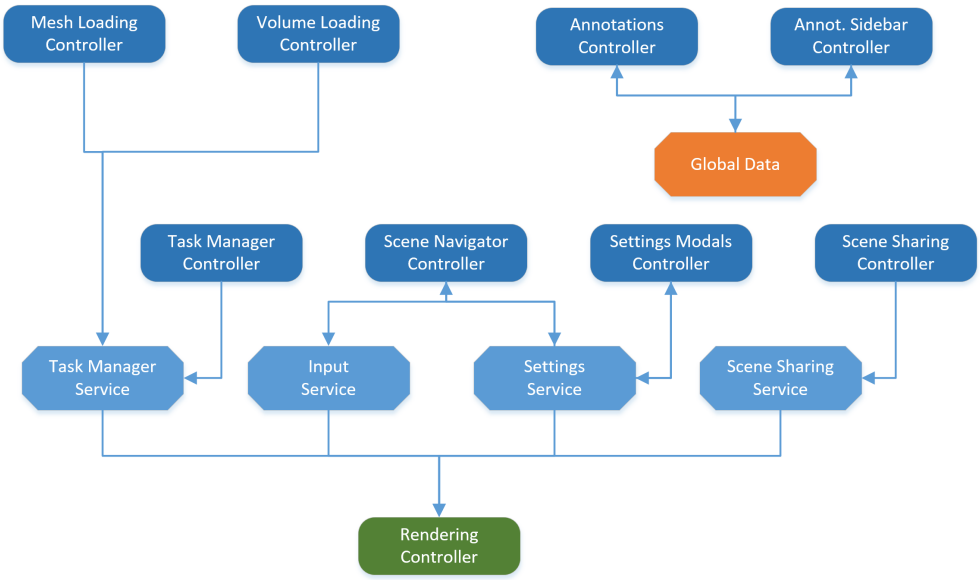
Slika 3.1: Slika prikazuje uporabniški vmesnik ogrodja Med3D.

Oddaljeno sodelovanje je v veliki meri omogočeno z zalednim delom ogrodja, gnanim na strežniku. Poleg oddaljenga sodelovanja nudi podporo tudi hrambi pogosto uporabljenih podatkov v obliki 3D mrežnih objektov ali skalarnih polj. Zaledni del sistema želimo v prihodnosti podpreti še s kompleksnejšimi algoritmi za obdelavo podatkov in oddaljenim upodablja-

njem. Tako bi v primeru zahtevnejših vizualizacij na slabših uporabniških napravah (mobilne naprave) te razbremenili z obdelovanjem oziroma upodabljanjem podatkov na strežniku.

3.1.1 Arhitektura sistema

Kot smo že omenili, ogrodje Med3D temelji na ogrodju AngularJS, ki vpeljuje dobre prakse razvoja in vzorce zasnove aplikacij. Temu primerno je razvito ogrodje sestavljeno iz gradnikov, ki jih izpostavlja AngularJS. Pri opisu arhitekture bomo zaradi splošnosti predpostavili, da so direktive vsebovane znotraj kontrolerjev. V nadaljevanju poglavja bomo predstavili različne gradnike ogrodja in njihove medsebojne odvisnosti, prikazane na sliki 3.2. V središču aplikacije se nahaja kontroler upodabljanja (angl. rendering controller), ki predstavlja najpomembnejši del ogrodja. Kot namiguje že ime, skrbi za upodabljanje, posodobitev scene in izris na platno, ki je vpeto čez celotno stran. V samo upodabljanje se bomo podrobneje spustili v naslednjih poglavjih.

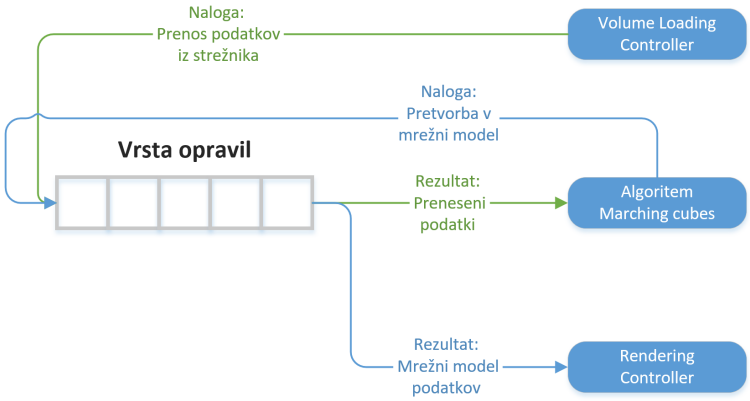


Slika 3.2: Slika prikazuje shemo arhitekture ogrodja Med3D.

Da lahko začnemo z upodabljanjem, je treba kontrolerju upodabljanja zagotoviti podatke, ki jih želimo vizualizirati. To naredimo s pomočjo kontrolerja za nalaganje mrež (angl. mesh loading controller) oziroma kontrolerja za nalaganje volumetričnih podatkov (angl. volume loading controller). Kontrolerja omogočata nalaganje mrežnih in volumetričnih podatkov, ki jih lahko lokalno naložimo iz naprave ali pa jih prenesemo iz Med3D strežnika. Na strežnik je treba podatke predhodno naložiti, kar nam ogrodje omogoča ob nalaganju lokalnih podatkov iz naprave. Ti se na strežniku shranijo v podatkovno bazo MongoDB [42], ki omogoča učinkovito shranjevanje in hitro poizvedovanje po podatkih.

Nalaganje velikih količin podatkov je pogosto zelo dolgotrajno, še posebej če jih želimo pred upodobitvijo tudi dodatno obdelati. To naredimo s pomočjo storitve za upravljanje nalog (angl. task manager service), ki omogoča tako asinhrono kot sinhrono izvajanje nalog, katerih napredek je prikazan na uporabniškem vmesniku s pomočjo kontrolerja za upravljanje nalog (angl. task manager controller). To omogoča boljšo uporabniško izkušnjo, saj ne blokira glavne niti, hkrati pa uporabniku grafično in tekstovno prikazuje dogajanje v ozadju ogrodja. Ko se naloga izvede do konca, so rezultati posredovani naslovljeni skupini naročnikov (angl. subscribers), ki je podana ob zagonu naloge. S primerom na sliki 3.3 je prikazano nalaganje volumetričnih podatkov iz strežnika in pretvorba naloženih podatkov v mrežni model z algoritmom Marching cubes. Tukaj gre za proces z dvema nalogama. Pri prvi se podatki naložijo iz strežnika in ob koncu nalaganja posredujejo algoritmu Marching cubes. Ta zažene drugo nalogo, ki pretvori volumetrične podatke v mrežni model in jih ob koncu izvajanja posreduje kontrolerju upodabljanja, ki jih začne upodabljati.

Poleg upodabljanja je ključnega pomena tudi interakcija z vizualizacijo oziroma sceno. V okviru ogrodja smo implementirali navigiranje s pomočjo tipkovnice in dveh grafičnih navigatorjev (zasuk in pomik). Za navigatorja skrbi kontroler scenskih navigatorjev (angl. scene navigation controller), ki beleži in pravilno interpretira miškine interakcije z navigatorji. Te podatke



Slika 3.3: Slika prikazuje izvajanje procesa, ki sestoji iz nalaganja volumetričnih podatkov in pretvorbe v mrežni model.

preko dvosmernega povezovanja podatkov (angl. two way data binding) deli s storitvijo za nadzor vhodnih enot (angl. input service). Ta je namenjena centraliziranju in združevanju vhodnih podatkov, ki prihajajo iz različnih virov, hkrati pa "spremlja" in interpretira pritiske tipk na tipkovnici. Kot smo že omenili, se je mogoče po sceni navigirati tudi s tipkovnico, s pomočjo tipk W, A, S, D, Q in E za zasuk ter smernih tipk in tipk R in F za pomik. Storitev ustrezno združuje transformacijske podatke tipkovnice in navigatorjev ter jih hrani javno dostopne vsem kontrolerjem. Tako lahko s kontrolerjem upodobitve ustrezno manipuliramo sceno glede na podatke, pridobljene s pomočjo vhodnih enot sistema.

Kot smo že omenili, ogrodje omogoča dodajanje tekstovnih anotacij na mrežni model. To smo realizirali s pomočjo kontrolerja anotacij (angl. annotation controller) in pomožne storitve z globalnimi podatki, ki pravzaprav predstavljajo kopico "singleton" objektov, od katerih je eden namenjen anotacijam. Kontroler pa je zadolžen za prikaz aktivnih anotacij in dodajanje novih. Vse anotacije se hkrati prikazujejo tudi v stranski vrstici, kar nam omogoča dodaten kontroler, ki poleg prikazovanja omogoča tudi odstranjevanje in skrivanje anotacij. V podrobnosti implementacije anotacij se bomo poglobili v naslednjih poglavjih tega dela.

Funkcionalnost delitve scene smo implementirali s pomočjo storitve in kontrolerja za delitev scene (angl. scene sharing controller, service). Storitev je tu zadolžena za komuniciranje s strežnikom, poslušanje po spremembah na vizualizirani sceni oziroma posodabljanje scene (v primeru, ko je gostitelj seje drug uporabnik ogrodja), sinhroniziranje annotacij in prilagajanje omrežni povezavi med strežnikom in gostujočo napravo. Delitev scene lahko do neke mere kontrolira uporabnik s pomočjo kontrolerja za delitev scene, hkrati pa z njim začne oziroma prekine oddaljeno sodelovanje z drugim uporabnikom ogrodja Med3D.

Vsi omenjeni kontrolerji in storitve pa so do neke mere nastavljivi. Te nastavitve konfiguriramo s pomočjo kontrolerja za nastavitve (angl. settings controller), ki omogoča odpiranje oken s konfiguracijskimi nastavitvami za različne dele ogrodja. Ob spremembah se podatki o nastavitvah dinamično posodabljajo in zapišejo v storitev za nastavitve (angl. settings service) s pomočjo dvosmernega povezovanja podatkov. Ta storitev je globalno na voljo vsem ostalim kontrolerjem, ki svoje delovanje prilagodijo glede na trenutno stanje nastavitev.

3.2 Upodabljanje

V tem podpoglavju bomo podrobneje predstavili delovanje omenjenega kontrolerja za upodabljanje in vseh pomožnih algoritmov, kot je na primer Marching cubes, ki jih v ogrodju uporabljamo za generiranje vizualizacie. Podrobneje bomo opisali upodobljevalnik mrežnih modelov (angl. renderer), ki je zadolžen za upodobitev podatkov. Predstavili bomo tudi organizacijo 3D prostora in vizualiziranih podatkov s pomočjo uporabe 3D objektov, scene in kamere. Opisali bomo tehnike neposrednega in posrednega upodabljanja, pomembne za vizualizacijsko ogrodje, ter njihovo implementacijo znotraj ogrodja Med3D. Za konec bomo predstavili še tehniko združevanja različnih tipov upodobitve s pomočjo upodabljanja na teksturo, ki spada pod tehnike odloženega upodabljanja (angl. deferred rendering).

Kakovost in učinkovitost upodobitve, ki jo je upodobljevalnik zmožen zagotoviti, sta močno pogojeni z organizacijo in strukturo vizualiziranih podatkov ter dobro razdelitvijo 3D prostora. Zato v ogrodju vse podatke vizualizacije predstavimo s hierarhičnim modelom v obliki drevesa t. i. graf scene, ki je sestavljeno iz 3D objektov. Ti objekti v osnovi definirajo svoj položaj in zasuk relativno na starša oziroma izhodišče koordinatnega prostora, če starša nimajo. Takšni osnovni objekti nam omogočajo razširitev njihovih funkcionalnosti in prilagoditev različnim namenom. V ogrodju smo tako definirali nekaj ključnih razširitev 3D objekta: objekt scene, kamere, luči, mreže in volumetričnih podatkov. Objekt scene predstavlja korenski element hierarhije, ki ga ob upodobitvi podamo upodobljevalniku skupaj z uporabljenim objektom kamere. Ta 3D objektu doda projekcijsko matriko, ki jo definira uporabljena projekcijska tehnika in njene nastavitve. Upodobljevalnik nato ob upodobitvi uporabi projekcijsko matriko kamere pri projiciranju scene na platno. Za pristno upodobitev so potrebni tudi objekti luči, s katerimi v 3D prostoru predstavimo vire svetlobe. Da je to mogoče, morajo luči razširiti 3D objekt z ustreznimi podatki, kot so barva, smer, pojenjanje in doseg. Za upodabljanje so nujno potrebni tudi podatki, ki jih v 3D prostoru predstavimo z mrežnimi objekti (angl. mesh) in volumni (angl. volume). Mrežni objekti so predstavljeni s pomočjo geometrije in materiala. Geometrija hrani različne podatke o ogliščih mreže, na primer položaje, normale, barve in teksturne koordinate, material pa vsebuje podatke o upodobitvenih lastnostih mreže. Pri volumnih geometrijo nadomesti skalarno polje, ki vsebuje podatke o položajih in intenzitetah vokslov.

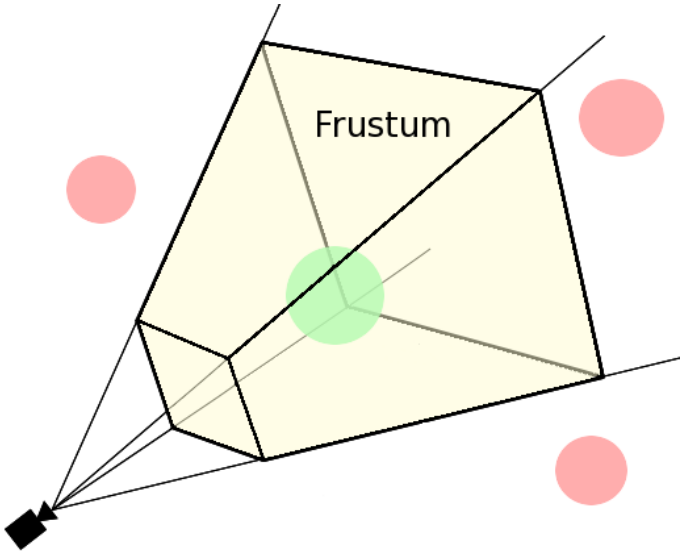
Poleg učinkovite strukture podatkov in dobre organizacije prostora h kakovosti upodabljanja močno prispevajo tudi številne funkcionalnosti in dobra implementacija upodobljevalnika, ki lahko zaradi zahteve po dinamičnosti in enostavnem nadgrajevanju predstavlja precejšen tehnični izziv. Upodobljevalnik, ki smo ga implementirali znotraj ogrodja Med3D, ni bil izjema. Ta poleg osnovnih funkcionalnosti upodobljevalnika obsega še številne naprednejše funkcionalnosti, kot so preskakovanje objektov, ki jih kamera ne vidi,

dinamično nalaganje senčilnikov iz strežnika, učinkovit način povezovanja podatkov s senčilniki, upodabljanje prosojnih objektov ter upodabljanje na teksturo.

Preskakovanje objektov, ki jih kamera ne vidi, smo realizirali s pomočjo uporabe tesno vsebujočih sfer mrežnih objektov. Za izračun tesno vsebujoče sfere si pomagamo z minimalno vsebujočo škatlo, ki definira položaj središča sfere. Nato se sprehodimo skozi vsa oglišča mreže objekta in določimo razdaljo med najbolj oddaljenim ogliščem in centrom sfere. Ta razdalja predstavlja nov radij tesno vsebujoče sfere. V najboljšem primeru je rezultat takšnega izračun sfera za $\sqrt{3}$ manjša od sfere, vpete okoli minimalne vsebujoče škatle. S pomočjo tesno vpetih sfer lahko objekte pred upodobitvijo testiramo, ali se sekajo s prostorom, ki ga vidi kamera (prirezana štiristrana piramida v primeru perspektivne projekcije oziroma kvader v primeru ortogonalne projekcije, angl. frustum). Tako lahko izločimo objekte, le-teh je lahko veliko, ki kameri niso vidni in ne vplivajo na končno upodobitev (glej sliko 3.4). S tem zmanjšamo število klicev izrisa in nepotrebnega procesiranja na grafičnem pospeševalniku.

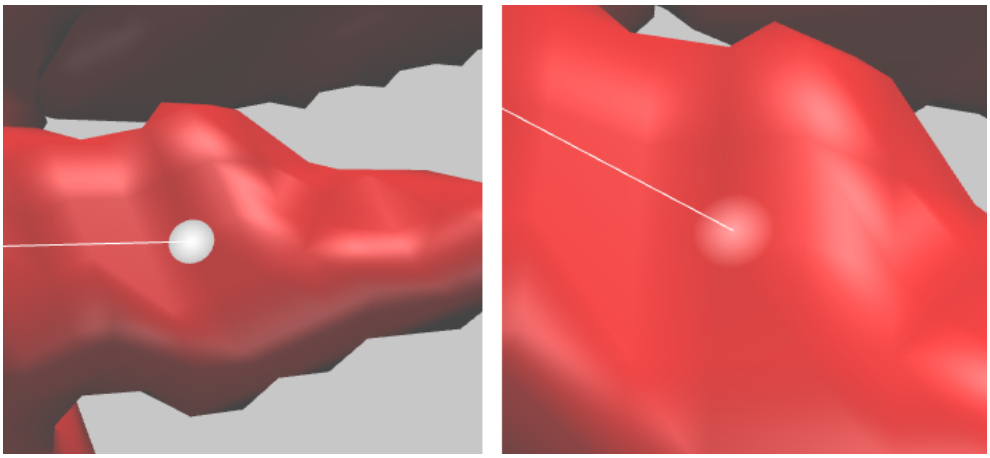
Dinamično nalaganje senčilnikov iz strežnika prispeva k hitrejšemu nalaganju ogrodja, saj se tako ob zagonu izognemo prenašanju in prevajanju velikega števila senčilnikov. Specifične senčilnike tako prenašamo šele, ko so ti nujno potrebni za upodobitev scene. Ti se prenesejo s pomočjo aplikacijskega programskega vmesnika XMLHttpRequest, ki s pomočjo HTTP zahtevka pridobi izvorno kodo senčilnika. Pridobljeno izvorno kodo nato prevedemo v obliko, primerno za uporabljeni grafični pospeševalnik, in posplošimo povezovanje podatkov, ki ga definira knjižnica WebGL, tako da generiramo Javascript funkcije, ki služijo učinkovitemu in enostavnemu povezovanju podatkov.

Za ogrodje je pomembno tudi upodabljanje prosojnih objektov, saj pogosto želimo prikazati tako objekte v ospredju kot tudi objekte v ozadju. Dober primer prosojnosti predstavljajo krogi, uporabljeni pri anotacijah za prikazovanje mesta pritrditve. V tem primeru želimo, da krogi dobro prikazujejo



Slika 3.4: Slika prikazuje preskakovanje objektov, ki niso v vidnem prostoru kamere. (http://karolyszanto.ro/MastersThesis/presentation/assets)

mesto pritrditve, hkrati pa želimo, da ne zastirajo pogleda uporabniku. To najlažje dosežemo z uporabo prosojnosti, kar realiziramo tako, da v senčilnik



Slika 3.5: Slika prikazuje pomanjkljivo (levo) in nepomanjkljivo (desno) implementacijo prosojnosti.

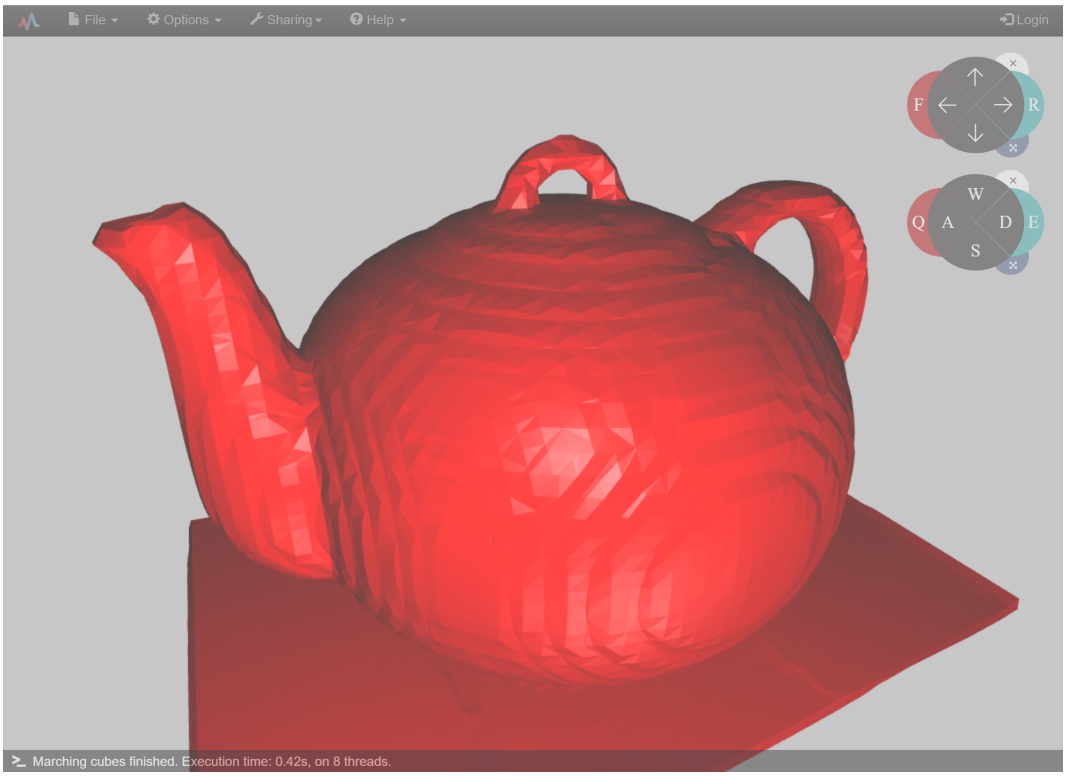
posredujemo dodatni podatek o prosojnosti oglišč oziroma kar celega objekta. Vendar to ni dovolj, saj lahko nepravilni vrstni red upodobitve objektov povzroči nepravilen izris v območju prosojnega objekta, kar prikazuje slika 3.5. Zato moramo najprej upodobiti neprosojne objekte in šele nato prosojne, ki jih uredimo padajoče glede na oddaljenost od kamere.

Vse omenjene funkcionalnosti smo v ogrodju uporabili za realizacijo posrednega upodabljanja volumetričnih podatkov.

3.2.1 Posredno upodabljanje

Razvito ogrodje nam s pomočjo osnovnega upodabljanja omogoča tudi posredno upodabljanje. Kot smo že omenili, pri posrednem upodabljanju volumetrične podatke v obliki skalarnega polja najprej pretvorimo v predstavitev s 3D mrežnim modelom in ga šele nato upodobimo. Za pretvorbo v ogrodju uporabljamo algoritem Marching cubes, saj velja za hiter, prostorsko učinkovit algoritem, ki ga je mogoče zelo dobro paralelizirati, hkrati pa nam vrača zadovoljive rezultate, katerih primer je prikazan na sliki 3.6. Tu je prostorska učinkovitost še posebej pomembna, saj nam spletni brskalnik omogoča dostop zgolj do omejenih virov, ki jih za naprednejše algoritme, kot je MPUI, ni dovolj. Poleg nizke prostorske zahtevnosti nam paralelizacija omogoča tudi do nekajkrat hitrejšo pretvorbo, saj pri procesiranju podatkov sinhronizacija in komunikacija med različnimi nitmi skoraj ni potrebna.

Za ogrodje Med3D smo razvili dve implementaciji algoritma Marching cubes. Prva je implementirana s pomočjo jezika Javascript, druga pa s pomočjo ASM.js [43], ki predstavlja nizkonivojsko podmnožico programskega jezika Javascript in dosega visoke pohitritve v primerjavi z osnovnimi implementacijami. Obe implementaciji sta uporabni tako za izvajanje v spletnem brskalniku kot tudi na strežniku v zalednem delu sistema, ki temelji na Node.js in ga programiramo s pomočjo jezika Javascript. Poleg omenjenih implementacij, ki temeljita na jeziku Javascipt, smo za primerjavo pri performančni analizi dodali tudi implementaciji v programskih jezikih C++ in Java. Rezultate performančne analize bomo podali in obrazložili v poglavju 3.4.



Slika 3.6: Slika prikazuje rezultat Marching cubes algoritma, izvedenega nad tridimenzionalnim skalarnim poljem objekta Utah teapot.

Implementaciji algoritma Marching cubes v programskem jeziku Javascript sta realizirani z uporabo aplikacijskega vmesnika Web Workers [44], ki omogoča asinhrono in paralelno izvajanje algoritma v brskalniku in na strežniku. Aplikacijski vmesnik pri tem izkorišča paralelnost centralne procesne enote in za vsako nit sproži lastno opravilo. Komunikacija s posameznim opravilom, ki se izvaja preko vmesnika Web Workers, poteka preko standardiziranega dogodkovno vodenega sporočilnega sistema. Poleg prenašanja sporočil med kopijami objektov omogoča tudi prenos objektov s prenosom njihovega lastništva. Posledično velikih objektov (skalarno polje) ni treba podvajati, ampak se na posamezno opravilo vmesnika Web Workers zgolj prenese njihovo lastništvo. S tem se izognemo dodatnemu delu in povečani porabi pomnilnika, do katere pride v primeru podvajanja objektov. Pomanj-

kanje pomnilnika lahko zaradi omejitev brskalnikov povzroči neodzivnost in nepredvideno prekinitev izvajanja.

Zaradi počasnejšega delovanja algoritma Marching cubes, implementiranega v programskem jeziku Javascript, kjer lahko pretvorba velikih skalarnih polj v mrežni model traja tudi več kot minuto, smo se odločili za uporabo programske knjižnice ASM.js. Programska knjižnica ASM.js dosega do desetkratno pohitritev izvajanja. Takšne pohitritve dosega zaradi uporabe prevajalnika AOT (angl. ahead-of-time), ki prevede in optimizira programsko kodo ob zagonu spletne aplikacije. Posledično se je treba odpovedati dinamičnosti programskega jezika Javascript in uporabiti zapis v obliki SSA (angl. static single assignment form). Zaradi nepraktičnosti takšnega zapisa se za generiranje programske kode ASM.js uporabljajo namenski prevajalniki. Le-ti ustrezno programsko kodo generirajo iz drugega nižjenivojskega programskega jezika, na primer C, C++. V našem primeru smo za osnovo uporabili programsko kodo, zapisano v programskem jeziku C++, in jo prevedli v obliko, primerno za uporabo s programsko knjižnico ASM.js, z uporabo prevajalnika Cheerp [45].

Glavni povod za izbiro prevajalnika Cheerp je pomnilniška organizacija, uporabljena v prevedeni programski kodi. Prevedena koda uporablja osnovno Javascript pomnilniško organizacijo, zato podatkov pred obdelavo ni treba dodatno pretvarjati oz. prenašati. To je bistvena prednost pred uporabo nekaterih drugih prevajalnikov, na primer prevajalnika Emscripten [46], ki uporablja lastno pomnilniško organizacijo, emulirano na pomnilniški organizaciji Javascripta.

Poleg algoritma Marching cubes želimo v prihodnosti podpreti tudi naprednejši, že omenjeni algoritem MPUI, ki omogoča kakovostnejšo pretvorbo volumetričnih podatkov. Tega zaradi visoke računske zahtevnosti in prostorske kompleksnosti ne bo mogoče realizirati v spletnem brskalniku, zato se bomo v tem primeru zanesli na zmogljivejši zaledni del na strežniku. Ta je omejen zgolj z uporabljeno strojno opremo, ki je ponavadi precej močnejša od uporabnikove strojne opreme. Pretvorba na strežniku bo hkrati omogočala

tudi hitrejši prenos modela k uporabniku, saj je predstavitev z mrežnim modelom prostorsko tudi do 100-krat manj zahtevna od predstavitve s skalarnim poljem.

Tehnike posrednega upodabljanja nam v ogrodju omogočajo kakovosten prikaz volumetričnih podatkov, vendar želimo uporabniku podati tudi nekoliko drugačen pogled na podatke, ki nam ga ponujajo pristopi neposrednega upodabljanja.

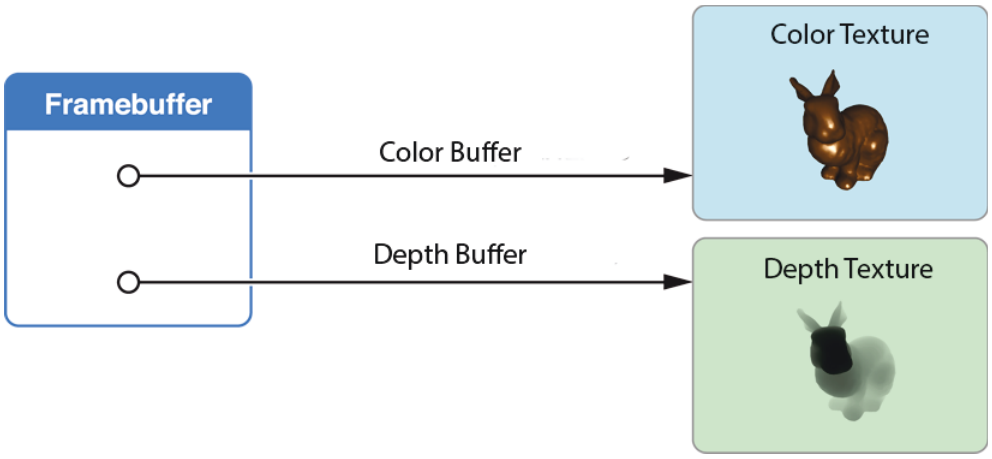
3.2.2 Neposredno upodabljanje

Kot smo že omenili, želimo poleg posrednega upodabljanja v ogrodju podpreti tudi različne tehnike neposrednega upodabljanja, specifično tehniko Splatting in metanje žarkov. Za realizacijo omenjenih metod bomo zasnovali upodobljevalnik za neposredno upodabljanje, ki bo razširjal trenutni upodobljevalnik mrežnih modelov. Za razširitev trenutnega upodobljevalnika smo se odločili, ker bo upodobljevalnik za neposredno upodabljanje lahko uporabljal velik delež že implementiranih funkcionalnosti, kot na primer dinamično nalaganje senčilnikov iz strežnika, učinkovit način povezovanja podatkov s senčilniki, upodabljanje na teksturo ter številne ostale metode za nastavitev knjižnice WebGL. Poleg trenutnega upodobljevalnika se bo lahko bodoči upodobljevalnik posluževal tudi že implementirane strukture podatkov, hierarhičnega razdeljevanja prostora in vseh manipulacij objektov v prostoru, ki jih ponuja struktura s 3D objekti.

Tehnike neposrednega upodabljanja nam podajo nov pogled na vizualizirane podatke, ki ima tako prednosti kot slabosti v primerjavi s pogledom, ki nam ga ponujajo tehnike posrednega upodabljanja. Zato smo v ogrodju omogočili združevanje različnih tipov upodobitev, s katerimi lahko združimo prednosti obeh pristopov.

3.2.3 Združevanje različnih tipov upodobitve

Z različnimi tipi upodobitve se pogosto pojavi tudi zahteva po združevanju izrisa. Različni tipi upodobitve namreč uporabniku v izrisu poudarijo različne lastnosti vizualiziranih podatkov. Združevanje takšnih upodobitev pogosto prispeva k boljši skupni predstavitvi podatkov, saj tako v končni upodobitvi zajamemo več lastnosti podatkov in uporabniku olajšamo njihovo interpretacijo, hkrati pa potrebujemo manj različnih vizualizacij za kakovostno interpretacijo podatkov. Najučinkovitejši način združevanja je izris različnih tipov upodobitev na teksture in nato združevanje različnih tekstur v končno upodobitev. Da je to mogoče, smo morali v upodobljevalniku implementirati odloženo upodabljanje oziroma upodabljanje na teksturo.



Slika 3.7: Slika prikazuje barvne in globinske podatke, ki jih ob upodobitvi zapišemo v teksturo.

Ob običajni upodobitvi se slika zapiše na rezerviran kos pomnilnika na grafičnem pospeševalniku (angl. framebuffer) in nato izriše na platno. Pri upodabljanju na teksturo moramo ta proces prilagoditi tako, da rezerviramo svoj kos pomnilnika, na katerega preusmerimo izris. Tako se ob upodobitvi slika ne bo izrisala na platno, ampak bo ta ostala zapisana na definiranem kosu grafičnega pomnilnika. Če želimo omenjeni kos uporabiti pri nadaljnjem upodabljanju, ga moramo pred upodabljanjem povezati tudi s teksturo, ki

jo nato pri izrisu uporabi knjižnica WebGL. V tako inicializirano teksturo se ob koncu izrisa iz grafičnega pospeševalnika prenese slika, ki je zapisana v rezerviranem kosu pomnilnika, kot je prikazano na sliki 3.7. Pridobljeno teksturo lahko sedaj uporabimo v naslednjem izrisu in jo poljubno združujemo z ostalimi teksturami s pomočjo senčilnikov. Poleg slike oziroma barvne informacije pa lahko tako pridobimo tudi informacijo o oddaljenosti od kamere (angl. z-buffer), ki nam za vsak piksel pove, kako daleč se nahaja najbližji objekt, ki je prispeval k njegovi upodobitvi. Ta informacija nam pomaga pravilno upoštevati prekrivanje različnih delov slike pri združevanju tekstur. Poleg združevanja izrisa nam takšno upodabljanje omogoča tudi naknadno obdelavo (angl. post-processing) oziroma apliciranje različnih filtrov, s katerimi lahko izboljšamo končno upodobitev.

3.3 Uporabniški vmesnik

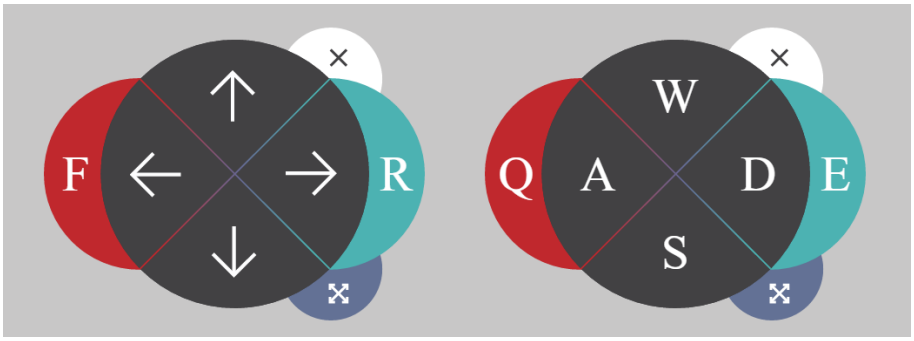
Pri vsakem vizualizacijskem ogrodju pomemben del predstavlja tudi uporabniški vmesnik, saj deluje kot povezava med uporabnikom in funkcionalnostmi ogrodja. S pomočjo vhodno-izhodnih naprav omogoča komunikacijo med uporabnikom in ogrodjem, s pomočjo katere lahko uporabnik nadzoruje delovanje ogrodja. Pri zasnovi uporabniškega vmesnika zato želimo omenjeno komunikacijo čim bolj poenostaviti, ne da bi s tem kakor koli okrnili funkcionalnosti ogrodja. Prav tako želimo, da ima uporabniški vmesnik lep izgled, ki je skladen z njegovim namenom, torej v našem primeru prikazovanje 3D medicinskih vizualizacij.

Ker smo ogrodje Med3D realizirali znotraj spletnega brskalnika, smo za zasnovo uporabniškega vmesnika uporabili gradnike oziroma elemente, ki jih definira standardiziran označevalni jezik za izdelavo spletnih strani HTML [47]. HTML sam po sebi predstavlja zgolj jezik za definiranje hierarhije gradnikov, njihovih podatkov in deloma funkcionalnosti ter ne omogoča modificiranja izgleda in položaja gradnikov. V ta namen se v kombinaciji s HTML uporablja slogovni jezik CSS, ki je namenjen definiranju izgleda HTML objek-

tov. Z uporabo HTML in CSS lahko tako zgradimo uporabniški vmesnik z želenim izgledom, ki ga je treba povezati z logiko vizualizacijskega ogrodja. To naredimo s pomočjo jezika Javascript, ki ima v brskalniku dostop do vseh HTML elementov in njihovih stilov. Ker pa HTML gradniki v osnovi zagotavljajo zgolj najbolj osnovne funkcionalnosti, se pri razvoju ogrodja pogosto zanesemo na pomožna ogrodja, kot je Bootstrap, ki ponujajo predloge gradnikov z naprednejšimi funkcionalnostmi. Ogrodje Med3D ni izjema. Za implementacijo uporabniškega vmesnika smo namreč uporabili številne Bootstrap predloge, ki smo jih dodatno razširili in prilagodili funkcionalnostim našega ogrodja. Poleg ogrodja Bootstrap smo uporabili tudi ogrodje Angular.js, ki nam omogoča definiranje lastnih gradnikov ter s tem boljšo strukturo in lažje razširljivo kodo. V nadaljevanju poglavja bomo podrobneje opisali različne gradnike uporabniškega vmesnika ogrodja Med3D: grafični navigatorji, navigacijska vrstica, stranska vrstica, okna za nalaganje podatkov in anotacije.

3.3.1 Grafični navigatorji

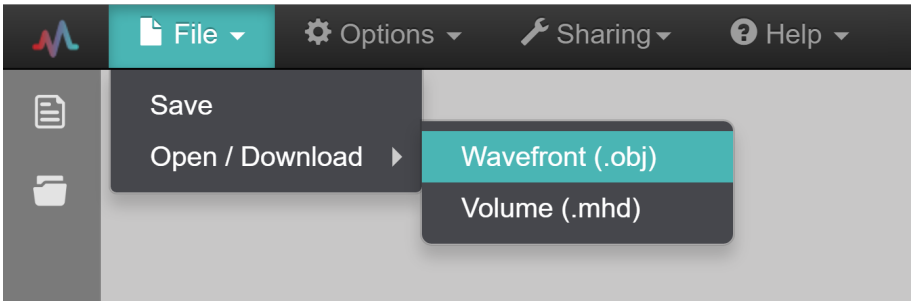
Grafična navigatorja v ogrodju predstavljata alternativo uporabi tipkovnice, saj omogočata navigiranje s pomočjo miške ali dotika. Posamezen navigator je implementiran s pomočjo vektorske grafike definirane v formatu SVG (angl. scalable vector graphics) [48], ki jo je mogoče v brskalniku uporabiti s pomočjo HTML5. Za navigiranje po sceni uporabljamo dva navigatorja, prikazana na sliki 3.8. Prvi omogoča pomik, drugi pa zasuk kamere vzdolž vseh osi relativno na trenutno orientacijo kamere. Navigator je sestavljen iz štirih glavnih delov: osrednjega navigacijskega kroga, stranskih navigacijskih krogov, držala, ki omogoča premikanje navigatorja po zaslonu (desno spodaj) in gumba za skrivanje navigatorja (zgoraj desno). Pri translacijskem navigatorju osrednji krog omogoča pomike po oseh Z (puščici naprej in nazaj) in X (puščici levo in desno), stranska kroga pa po osi Y. Pri rotacijskem osrednji krog omogoča zasuk okoli osi X (W in S) in Y (A in D), stranska kroga pa okoli Z osi.



Slika 3.8: Slika prikazuje translacijski (levo) in rotacijski (desno) grafični navigator scene.

3.3.2 Navigacijska vrstica

Navigacijska vrstica, kot namiguje že ime, služi za navigiranje po različnih funkcionalnostih ogrodja. Z njeno pomočjo dostopamo do različnih oken, kot so na primer okno za nalaganje mrežnih objektov, okno za nalaganje volumetričnih podatkov, okno nastavitev ter gumb za začetek oddaljenega sodelovanja.



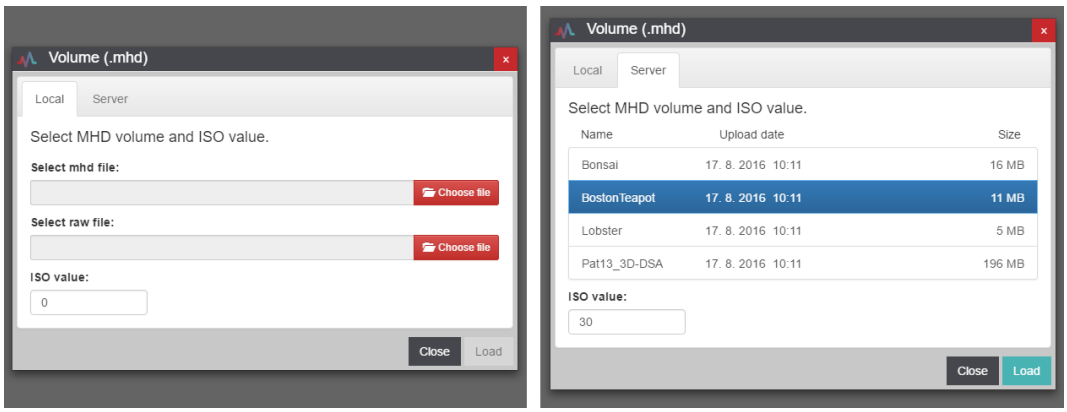
Slika 3.9: Slika prikazuje navigacijsko vrstico z odprtim dvostopenjskim spustnim menijom.

Implementirali smo jo s pomočjo Bootstrapove predloge, imenovane "Navbar", ki v osnovi podpira gumbe in enonivojske spustne menije. Za namen ogrodja smo osnovno predlogo stilsko preoblikovali, tako da je ta skladna z

ogrodjem, in implementirali možnost večstopenjskega spustnega menija, prikazanega na sliki 3.9. Poleg omenjenih razširitev smo orodno vrstico tudi konfigurirali, tako da ob neuporabi delno zbledi in tako uporabniku predstavlja manjšo distrakcijo.

3.3.3 Okna

Ker je ogrodje Med3D implementirano kot enostranska spletna aplikacija, je treba uporabniku vsebino uporabniškega vmesnika prikazovati na dinamičen način. Zato smo se odločili, da bomo v ogrodju vsebino prikazovali s pomočjo Bootstrapovih objektov, imenovanih "Modal", ki predstavljajo dinamična okna, ki jih je mogoče prosto premikati po zaslonu. Takšen način prikaza na nek način posnema prikaz vsebine v operacijskih sistemih, kar omogoča uporabnikom hitro spoznavanje s funkcionalnostmi oken.



Slika 3.10: Slika prikazuje okna za nalaganje volumetričnih podatkov iz lokalne naprave (levo) ter iz strežnika (desno).

Na sliki 3.10 je prikazano eno izmed najpogosteje uporabljenih oken, ki se uporablja za izbiro volumetričnih podatkov, ki jih želimo naložiti v ogrodje. To okno je lahko prikazano v dveh različnih stanjih, ki ju definiramo s pomočjo uporabe Bootstrapove predloge "Nav", ki omogoča menjavanje pogledov znotraj okna. Prvo se uporablja za nalaganje lokalnih podatkov

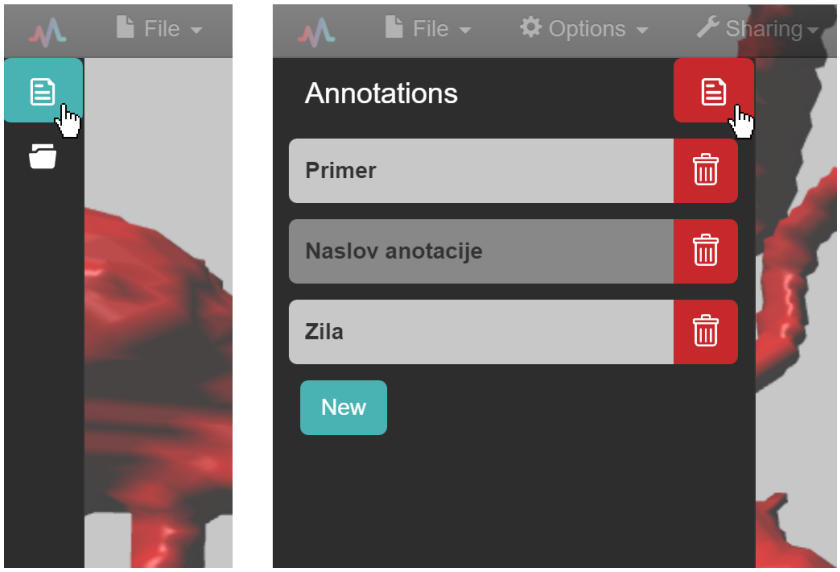
in omogoča izbiro datotek na uporabljani lokalni napravi. Drugo pa omogoča izbiro in prenos volumetričnih podatkov, navedenih na prikazanem seznamu, ki se asinhrono osveži s podatki iz strežnika vsakič, ko odpremo okno.

3.3.4 Stranska vrstica

Poleg navigacijske vrstice, ki skrbi za navigiranje po različnih funkcionalnostih, smo implementirali tudi stransko vrstico, ki je namenjena orodjem in metodam za interakcijo z vizualizacijo. Za te želimo, da so čim hitreje dostopne iz glavnega pogleda, saj predstavljajo največji del interakcije z uporabnikom. Zato smo se odločili za uporabo stranske vrstice, ki je uporabniku ves čas dostopna na levi strani zaslona, hkrati pa ne zaseda veliko prostora, saj jo je mogoče zložiti. Slika 3.11 prikazuje stransko orodno vrstico v zaprtem (levo) in odprtem stanju (desno), v katerem je izbrano orodje za nadzorovanje anotacij. Za stranske vrstice Bootstrap ne ponuja predloge, tako da smo jo morali oblikovati, animirati in zasnovati njeno logiko v celoti. V trenutni implementaciji omogoča prikaz poljubno mnogo orodij v zaprtem stanju, v odprtem stanju pa prikaže poljubno vsebino, ki je povezana z izbranim orodjem.

3.3.5 Podpora uporabniškim anotacijam

Za konec poglavja o uporabniškem vmesniku bomo predstavili še podporo uporabniškim anotacijam, ki veljajo za najkompleksnejši del uporabniškega vmesnika, saj je treba del anotacij upodobiti na platnu, del pa je prikazan v kontekstu HTML dokumenta. Kot smo že omenili, vse podatke v zvezi z anotacijami hranimo v "singleton" objektu, ki je dostopen vsem delom uporabniškega ogrodja. To nam omogoča, da anotacije realiziramo s pomočjo različnih delov ogrodja: orodja za anotacije v stranski orodni vrstici, kontrolerja anotacij in kontrolerja za upodabljanje. Orodje za anotacije, prikazano skrajno levo na sliki 3.12, nam omogoča pregled nad vsemi anotacijami, dodajanje novih anotacij, njihovo skrivanje in brisanje. Podatki v orodju so pove-



Slika 3.11: Slika prikazuje stransko orodno vrstico v zaprtem (levo) in odprtem stanju (desno).

zani neposredno z anotacijskim objektom (singleton) s pomočjo dvosmernega podatkovnega povezovanja (angl. two way data binding), kar nam omogoča, da se vse spremembe, ki jih naredimo z orodjem, odražajo na globalnem stanju in obratno. Kot primer lahko vzamemo dodajanje nove anotacije. To začnemo s pomočjo orodja za anotacije, ki ob zahtevi po novi anotaciji zgolj ustvari prototip podatkov, nato pa kontroler anotacij zazna spremembo v podatkih in ustrezno ustvari okno za vnos nove anotacije, prikazano na sredini slike 3.12.

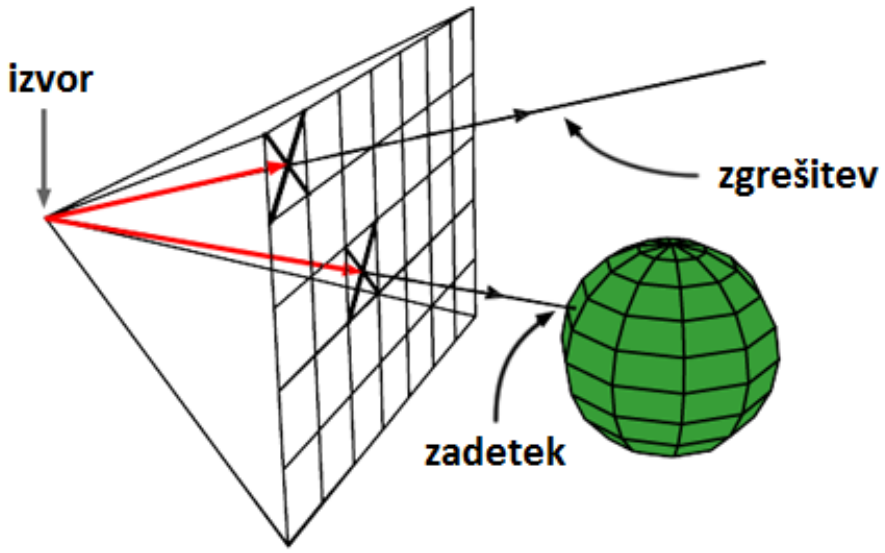
Kontroler anotacij deluje v tesni povezavi s kontrolerjem za upodabljanje. Kontroler anotacij skrbi za del anotacij, prikazan v kontekstu HTML, kar vključuje vnos tekstovne vsebine in zaključevanje generacije nove anotacije ter prikaz in posodabljanje podatkov o položaju ter dimenzijah vseh anotacijskih oken. Ta je namreč mogoče poljubno premikati znotraj ogrodja in spreminjati njihovo velikost. Kontroler anotacij nam tako v kombinaciji z orodjem za anotacije omogoča prikaz, dodajanje, skrivanje in brisanje anotacijskih oken. Poleg omenjenih funcionalnosti pa želimo uporabniku omogočiti

Slika 3.12: Slika prikazuje tri aktivne anotacije, stransko orodno vrstico z odpritim orodjem za urejanje anotacij (levo) in okno za ustvarjanje nove anotacije (sredina).

tudi povezovanje anotacij z izrisanimi objekti na platnu, za kar poskrbi kontroler upodabljanja.

Povezovanje anotacije z izrisanimi objekti je mogoče v fazi dodajanja anotacije z uporabo miške. Z miško kliknemo na izrisan del objekta, s katerim želmo anotacijo povezati. Ob kliku se na objekt pritrdi krog, ki nakazuje točko interesa in se s črto poveže na okno anotacije, ki jo trenutno gradimo. Da je to mogoče, je treba položaj klika na platnu preslikati na ustrezno točko najbližjega presekanega objekta. To naredimo s pomočjo iskanja preseka z žarkom (angl. Ray casting). V prvem koraku tega procesa moramo skonstruirati žarek, sestavljen iz vektorja $\vec{o}$, ki predstavlja položaj izvora žarka, in enotskega vektorja $\vec{d}$, ki predstavlja smer žarka. Pri perspektivni projekciji položaj izvora žarka predstavlja kar položaj kamere v prostoru, smer žarka pa izpeljemo kot $\vec{d} = normalize(M * P^{-1} * \vec{k} - \vec{o})$, kjer M predstavlja matriko, ki preslika kamero iz izhodišča koordinatnega sistema v trenutni položaj, P^{-1}

inverz projekcijske matrike in $\vec{k}$, ki predstavlja točko klika. Pri ortogonalni projekciji pa izvor žarka izpeljemo kot $\vec{o} = M * P^{-1} * \vec{k}$, kjer vektor položaja klika $\vec{k}$ nekoliko prilagodimo, tako da X in Y koordinati predstavljata točko klika, Z koordinata pa je definirana kot $z = \frac{near+far}{near-far}$, kjer $near$ predstavlja razdaljo do sprednje ravnine, ki omejuje pogled, far pa razdaljo do zadnje. Smer žarka definiramo kot $\vec{d} = [0, 0, -1]^T$. V drugem koraku iskanja preseka s tako zgrajenim žarkom poiščemo presek z najbližjim objektom znotraj vidnega prostora kamere (glej sliko 3.13). Z dobljenim presekom lahko nato definiramo povezovalno črto med točko interesa in oknom anotacije.



Slika 3.13: Slika prikazuje iskanje preseka z žarkom.

3.4 Performančna analiza sistema

V tem poglavju bomo predstavili analizo performančno zahtevnejših operacij, ki jih izvaja ogrodje. Te obsegajo hitrost mrežnega upodabljanja z in brez osvetljevanja, hitrost kontrolnega dela upodobljevalnika, ki upravlja s knjižnico WebGL ter kontrolira izris scene in algoritma Marching cubes, ki

ga uporabljamo za pretvorbo iz volumetričnih v mrežne podatke. Vse analize smo izvedli na sistemu z naslednjimi specifikacijami:

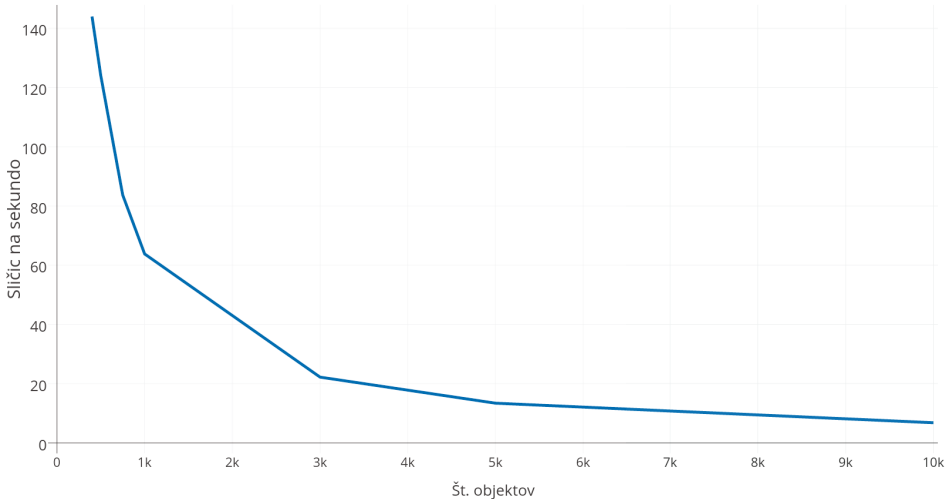
- Procesor: Štirijederni procesor Intel i7 6700K (4.2 GHz),
- Pomnilnik: 16 GB DDR4 (2400 MHz),
- Operacijski sistem: Windows 10,
- Grafični pospeševalnik: Gigabyte GeForce GTX 980 Ti RAM: 6 GB.

Za vse spletne implementacije pa smo uporabili 64-bitni brskalnik Chrome (različica 51). Celotna analiza je razdeljena na dve podpoglavji. Prvo je 3.4.1, ki poda analizo upodobljevalnika, drugo pa 3.4.2, ki predstavi analizo različnih implementacij algoritma Marching cubes.

3.4.1 Upodabljanje

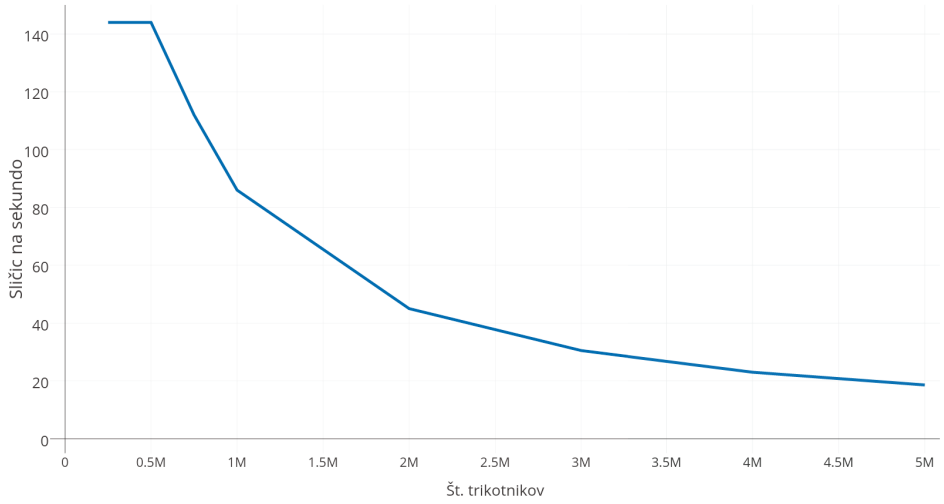
Kot smo že omenili, smo v sklopu upodabljanja analizirali hitrost mrežnega upodabljanja in hitrost kontrolnega dela upodobljevalnika, ki upravlja s knjižnico WebGL in pripravi podatke scene za izris. Pri analizi hitrosti mrežnega upodabljanja smo testirali tako izris z osvetljevanjem, kjer smo uporabili Phongov model, kot tudi izris brez osvetljevanj in primerjali hitrost izrisa, podano s sličicami na sekundo v relaciji s številom izrisanih trikotnikov. Teste smo izvedli v kontroliranem okolju, kjer smo na sceno postavili objekt z N naključno postavljenimi enakostraničnimi trikotniki s stranico dolgo 10 enot in jo upodobili s pomočjo ortogonalne projekcije, da so vsi trikotniki ohranili svojo velikost na platnu velikosti $1920 * 1080$. Oba izrisa smo testirali z 0,25, 0,5, 0,75, 1, 2, 3, 4 in 5 milijoni trikotnikov, pri izrisu s Phongovim osvetljevanjem pa smo uporabili tudi deset scenskih luči, izmed tega pet usmerjenih in pet točkovnih. Rezultati testiranja izrisa brez osvetljevanja so prikazani z grafom na sliki 3.15, z osvetljevanjem pa z grafom na sliki 3.16. Zaradi uporabe vertikalne sinhronizacije (angl. vertical synchronization – V-Sync), ki navzgor omeji hitrost izrisa s frekvenco zaslona (v

našem primeru 144 Hz), so naši rezultati navzgor omejeni s 144 sličicami na sekundo. Kot lahko razberemo z grafov, z upodabljanjem brez osvetljevanja dosežemo približno dvajset izrisov na sekundo več kot z upodabljanjem z osvetljevanjem.

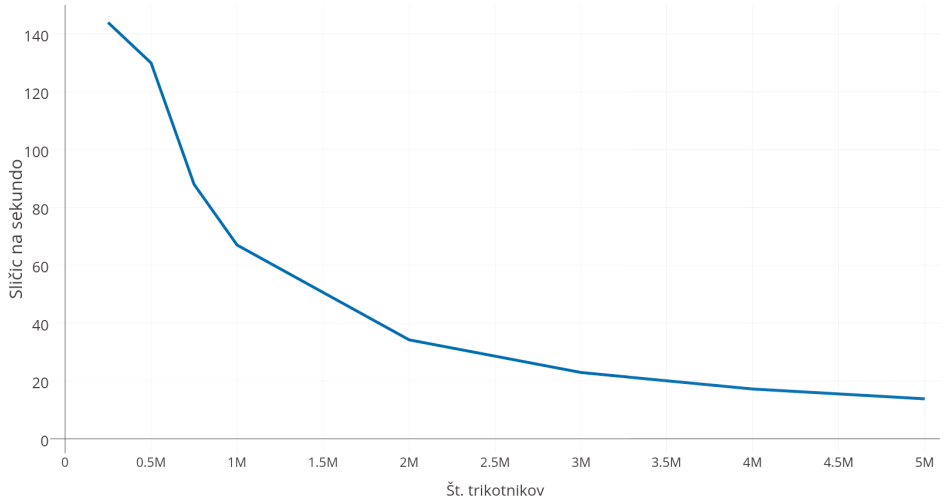


Slika 3.14: Slika prikazuje hitrost upodabljanja v relaciji s številom objektov na sceni.

Testiranje kontrolnega dela upodobljevalnika smo izvedli tako, da smo tako kot pri testiranju mrežnega upodabljanja generirali N trikotnikov, vendar smo v tem primeru vsak trikotnik zapakirali v svoj objekt. S tem močno obremenimo kontrolni del upodobljevalnika, ki skrbi za pripravo objektov za izris. Testirali smo izris s 400, 500, 750, 1000, 3000, 5000 in 10000 objekti in hitrost izrisa predstavili s sličicami na sekundo v relaciji s številom objektov. Rezultati testiranja so prikazani z grafom na sliki 3.14. Z grafa lahko razberemo, da začne hitrost izrisa s 500 objekti strmo padati saj dosežemo maksimano obremenitev jedra procesorja, ki ga uporablja kontrolni del upodobljevalnika.



Slika 3.15: Slika prikazuje hitrost upodabljanja brez uporabe osvetljevanja v relaciji s številom upodobljenih trikotnikov.



Slika 3.16: Slika prikazuje hitrost upodabljanja z uporabo osvetljevanja (Phong) v relaciji s številom upodobljenih trikotnikov.

3.4.2 Marching cubes

V preformančno analizo algoritma Marching cubes smo vključili štiri implementacije: Java, C++, Javascript in ASM.js. Spletni implementaciji Javascript in ASM.js v spletnem brskalniku Chrome. Kot vhodne podatke smo uporabili enakostranične volumne v treh razsežnostih (128^3, 256^3 in 512^3), generirane z enačbo enodelnega hiperboloida $vol_{xyz} = x^2 + y^2 - z^2 - 25$, in realni primer vratnih žil, velikosti $512 \times 512 \times 390$. Realni primer smo testirali s tremi vrednostmi praga: nizko, srednjo in visoko, pri katerih dobimo mrežne modele s približno 6, 1 in 0,25 milijona trikotnikov. Vse omenjene implementacije omogočajo tudi izkoriščanje vzporednega izvajanja, zato smo poleg zaporednega testa v eni niti hitrost izmerili tudi pri uporabi dveh in štirih niti. Vsaka konfiguracaija je bila ovrednotena s povprečnim rezultatom desetih iteracij. Dobljen povprečni rezultat pa smo ustrezno zaokrožili glede na standardni odklon.

Implementacije, ki omogočajo prevajalniško optimizacijo (C++, ASM.js), so bile za testiranje prevedene z optimizacijo za čim hitrejše izvajanje. Poleg C++ in ASM.js pa tudi Java omogoča optimizacije, vendar šele med izvajanjem. Te realizira s pomočjo sprotnega prevajalnika, ki pogosto izvedeno vmesno kodo prevede v domorodno obliko. Zaradi tega smo pri testiranju implementacije v Javi to najprej "ogreli", tako da smo pred merjenjem časa izvedli dve iteraciji algoritma, ki nista bili upoštevani v rezultatih.

Rezultati ovrednotenja so prikazani v tabelah 3.1 in 3.2. Kot bi pričakovali, je iz rezultatov razvidno, da je v vseh primerih najhitrejša implementacija v programskem jeziku C++. Pri umetno generiranih podatkih se ji pri večjih velikostih povsem približa implementacija v programskem jeziku Java. Kljub temu, da se omenjenima implementacijama pričakovano ne približa nobena implementacija v programskem jeziku Javascript, je iz rezultatov razvidno, da je uporaba programske knjižnice ASM.js smiselna, saj doseže na umetno generiranih podatkih tudi več kot osemkratno pohitritev, na izbranem primeru realnih podatkov pa do 1,5-kratno pohitritev izvajanja.

V poglavju smo predstavili performančno analizo upodabljanja in pre-

Vol. Size	C++	Java	Javascript	ASM.js
1 nit				
128^3	0,02 s	0,03 s	0,66 s	0,16 s
256^3	0,11 s	0,21 s	4,85 s	0,98 s
512^3	0,79 s	1,32 s	38,00 s	4,63 s
2 niti				
128^3	0,01 s	0,03 s	0,38 s	0,11 s
256^3	0,06 s	0,09 s	2,70 s	0,61 s
512^3	0,41 s	0,76 s	21,80 s	3,95 s
4 niti				
128^3	0,01 s	0,01 s	0,26 s	0,08 s
256^3	0,04 s	0,05 s	1,60 s	0,42 s
512^3	0,27 s	0,47 s	12,50 s	2,42 s

Tabela 3.1: Rezultati performančne analize izvajanja algoritma Marching cubes na generiranih podatkih pri različnih stopnjah vzporednega računanja.

tvorbe volumetričnih podatkov v mrežni model. Z dobljenimi rezultati smo zadovoljni, saj so vsi v mejah pričakovanj ali te celo presegajo. Prav tako so nam analize pomagale sprejemati pomembne implementacijske odločitve. Tu še posebej izstopa analiza algoritma Marching cubes, saj smo se zaradi dobljenih rezultatov odločili za implementacijo s pomočjo ASM.js.

Prag	C++	Java	Javascript	ASM.js
1 nit				
Nizek	0,80 s	1,50 s	22,20 s	15,30 s
Srednji	0,57 s	0,91 s	20,70 s	15,20 s
Visok	0,52 s	0,85 s	20,20 s	15,10 s
2 niti				
Nizek	0,47 s	0,70 s	12,30 s	8,90 s
Srednji	0,30 s	0,46 s	11,50 s	8,70 s
Visok	0,28 s	0,44 s	11,20 s	8,70 s
4 niti				
Nizek	0,34 s	0,42 s	8,70 s	5,80 s
Srednji	0,19 s	0,29 s	7,50 s	5,80 s
Visok	0,17 s	0,26 s	7,40 s	5,50 s

Tabela 3.2: Rezultati performančne analize izvajanja algoritma Marching cubes na realem primeru podatkov pri različnih stopnjah vzporednega računanja.

Poglavje 4

Oddaljeno sodelovanje

Pri vizualizaciji podatkov se velikokrat pojavi potreba po delitvi in skupni interpretaciji vizualizacij z oddaljenimi osebami. Na primeru medicine to pogosto pomeni pridobivanje mnenj oddaljenih specialistov. Delitev podatkov, lokalno vizualiziranje ter intepretacija so pogosto precej neučinkoviti tako z vidika časovne potratnosti kot tudi z vidika prenosa informacij med uporabniki. Ta problem smo v predstavljanem vizualizacijskem ogrodju rešili z implementacijo oddaljenega sodelovanja, ki jo bomo v tem poglavju podrobneje predstavili.

Začeli bomo z opisom delitve scene, ki predstavlja najpomembnejši del oddaljene interakcije, saj omogoča učinkovito delitev vizualiziranih podatkov in njihove strukture z majhno porabo internetne pasovne širine. V nadaljevanju bomo prodobneje opisali klepet, ki je podprt znotraj ogrodja in omogoča tekstovno komunikacijo med povezanimi uporabniki. Predstavili bomo tudi delitev že omenjenih anotacij ter njihov prispevek k hitrejši interpretaciji podatkov. Za konec bomo opisali performančno analizo komunikacije ter podali rezultate in jih ovrednotili.

4.1 Delitev scene

Glavna funkcionalnost oddaljenega sodelovanja je delitev scene, ki omogoča delitev podatkov v obliki scenskega grafa ter realnočasovno sinhroniziranje gradnikov grafa in pogleda kamere med uporabniki. Ti lahko z ustreznimi dovoljenji prevzamejo nadzor nad kamero, kar omogoča dodatno prostost in več dinamičnosti med skupno interpretacijo vizualizacije. Takšna delitev vizualiziranih podatkov omogoča dobro podlago za realizacijo kakovostnega oddaljenega sodelovanja, saj omogoča bistveno hitrejšo interpretacijo vizualizacije ter prenos informacij in znanja med uporabniki kot lokalno vizualiziranje in interpretacija podatkov.

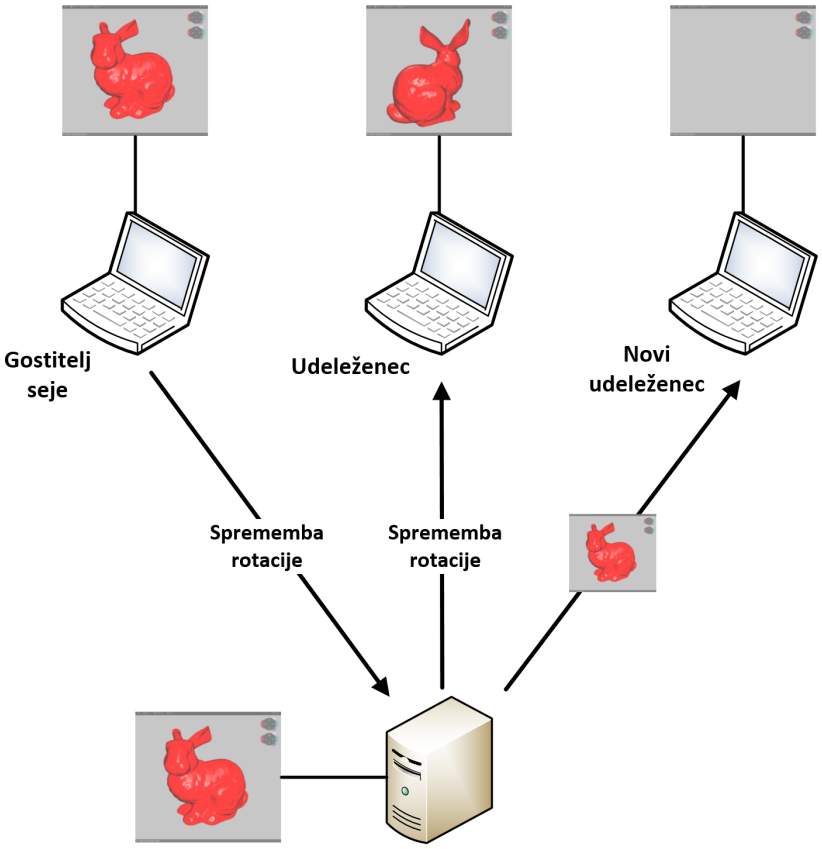
Ko uporabnik vzpostavi želeno sceno, jo lahko prične deliti z ostalimi uporabniki vizualizacijskega ogrodja. Ob inicializaciji oddaljenega sodelovanja se kamera, scena in vsi objekti v njeni hierarhiji izvozijo v JSON objekt, ki ga sestavljajo seznami metapodatkov objektov, materialov in geometrij mrežnih objektov. Seznam metapodatkov objektov za vsak izvožen objekt vsebuje položaj, zasuk, skalo ter identifikator starša. Izvožen objekt je opremljen tudi s pripadajočim unikatnim identifikatorjem, ki nam omogoča, da lahko na prejemnikovi strani pravilno rekonstruiramo graf scene. Mrežne objekte poleg metapodatkov sestavljata tudi geometrija in material, ki se izvozita v seznam geometrij in seznam materialov. V seznam geometrij se za vsako geometrijo izvozijo podatki, kot so položaji, normale, indeksi in barve oglišč, v seznam materialov pa se za vsak material izvozijo podatki, kot na primer barva, nastavitve upodobitve, prosojnost in faktor odsevnosti. Vse izvožene geometrije in materiali so tako kot objekti opremljeni z unikatnim identifikatorjem, kar nam omogoča, da jih na prejemnikovi strani povežemo z mrežnim objektom, ki mu pripadajo. Da je izvoz hierarhije v JSON objekt možen, morajo vsi objekti, geometrije in materiali omogočati izvoz svojih lokalnih podatkov v zapis JSON. Tako zapakirani podatki se nato preko protokola WebSocket prenesejo na strežnik, preko katerega poteka tudi vsa nadaljnja komunikacija. Strežnik ob prejemu podatkov te shrani in odpre novo sejo oddaljenega sodelovanja. Ustvarjeni seji se lahko pridružijo vsi

ostali uporabniki ogrodja, ki jim je bila dodeljena pravica za dostop do seje. Po končani inicializaciji oddaljenega sodelovanja se začne faza sinhronizacje. V fazi sinhronizacije začne gostitelj seje oddajati vse spremembe, ki jih naredi na deljenih objektih, grafu scene in kameri, hkrati pa mora posredovati tudi podatke o vseh na novo dodanih objektih. Frekvenca pošiljanja sprememb se dinamično spreminja glede na kakovost povezave med gostiteljem in strežnikom, saj bi v primeru prepogostega pošiljanja in slabe kakovosti pošiljanja prišlo do visokega časovnega zamika ali celo nepričakovane napake v protokolu WebSocket. Ko strežnik prejme paket s spremembami, najprej popravi svojo različico podatkov, nato pa jih posreduje vsem uprabnikom, ki so udeleženi v seji oddaljenega sodelovanja. Ker vsa komunikacija poteka preko strežnika, je gostitelj seje močno razbremenjen, saj mora posodobitve pošiljati samo strežniku, ta pa nato poskrbi, da vsi udeleženci dobijo svojo kopijo podatkov, hkrati pa ob udeležbi novega uporabnika gostitelju ni treba zopet pošiljati vseh podatkov, ampak jih lahko nov uporabnik prenese kar s strežnika.

Shema komunikacije pri delitvi scene je prikazana na sliki 4.1, kjer je prikazano, kako poteka sinhronizacija scene med dvema povezanima uporabnikoma na primeru spremembe orientacije kamere in potek priklopa novega uporabnika, ki prejme celoten opis scene. Takšna implementacija delitve scene omogoča zelo odzivno interakcijo z objekti v seji in ponuja enak vpogled v podatke več uporabnikom ogrodja sočasno in z minimalnim zamikom. Za lažjo interpretacijo tako deljene scene v sodelovanju z ostalimi uporabniki smo v ogrodju omogočili tudi uporabo klepeta.

4.2 Klepet

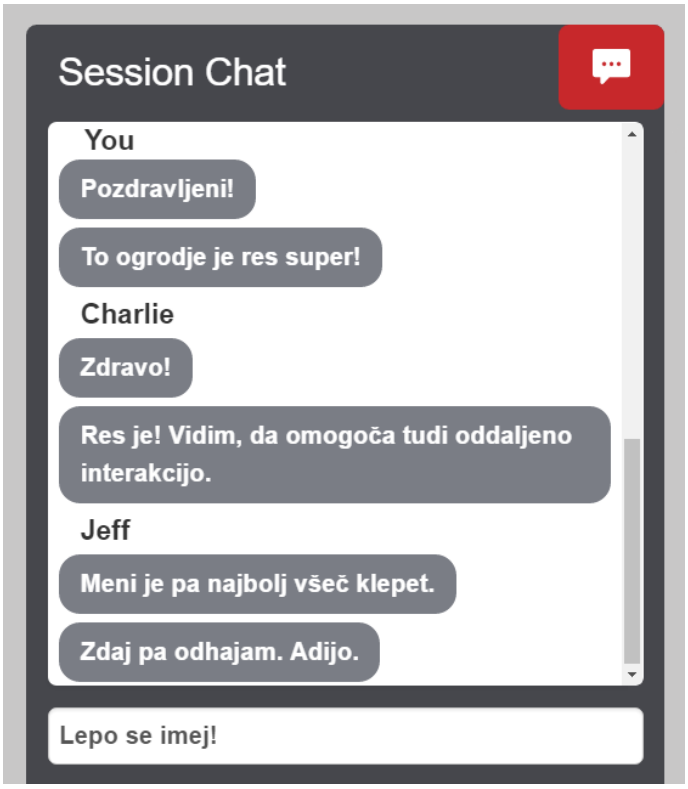
Poleg delitve scene ogrodje v okviru oddaljenega sodelovanja omogoča tudi klepet med uporabniki, udeleženimi v isti seji. Klepet uporabnikom nudi tekstovno komunikacijo v obliki kratkih sporočil, prikazano na sliki 4.2, ki jih lahko uporabniki pošiljajo z visoko frekvenco in so podpisana z upo-



Slika 4.1: Slika prikazuje shemo komunikacije pri oddaljenem sodelovanju. Levo zgoraj je gostitelj seje, ki deli sceno z ostalimi uporabniki spletne aplikacije. Gostitelj je na shemi že sinhroniziral sceno s strežnikom (spodaj desno) in mu posreduje posodobitev deljene scene, ki jo strežnik razpošlje vsem naročnikom in posodobi svojo lokalno kopijo scene. Zgoraj na sredini je prikazan udeleženec seje, ki je že prenesel sceno iz strežnika in sedaj prejema posodobitve gostitelja. Desno zgoraj pa je nov udeleženec seje, ki iz strežnika prenaša zadnjo verzijo scene.

rabniškim imenom pošiljatelja. Deloma se zgleduje po klepetih različnih socialnih omrežij, ki omogočajo zgolj kratka tekstovna sporočila. Klepet poleg sporočil izpisuje tudi informacije o prihodu in odhodu udeležencev

seje. Tako kot delitev scene je realiziran s pomočjo protokola WebSocket, pravzaprav uporablja kar isto vtičnico (angl. socket). Ko uporabnik pošlje sporočilo, se to preko omenjenega vtiča najprej posreduje na strežnik, ki sporočilo razpošlje vsem uporabnikom seje, v kateri je udeležen pošiljatelj. Da je to mogoče, mora strežnik voditi evidenco o vseh uporabnikih, povezanih na sejo, ki jim poleg sporočil razpošilja tudi informacije o prihajanju in odhajanju uporabnikov. Klepet se v ogrodju prikaže v spodnjem desnem kotu, ko se udeležimo seje oziroma jo gostimo. Ker lahko prikazan klepet moti uporabnikov pogled, ga je mogoče skriti oziroma prikazati z namensko tipko. Tako implementiran klepet bistveno poenostavi komunikacijo med uporabniki in omogoča dober pregled nad udeleženci seje.



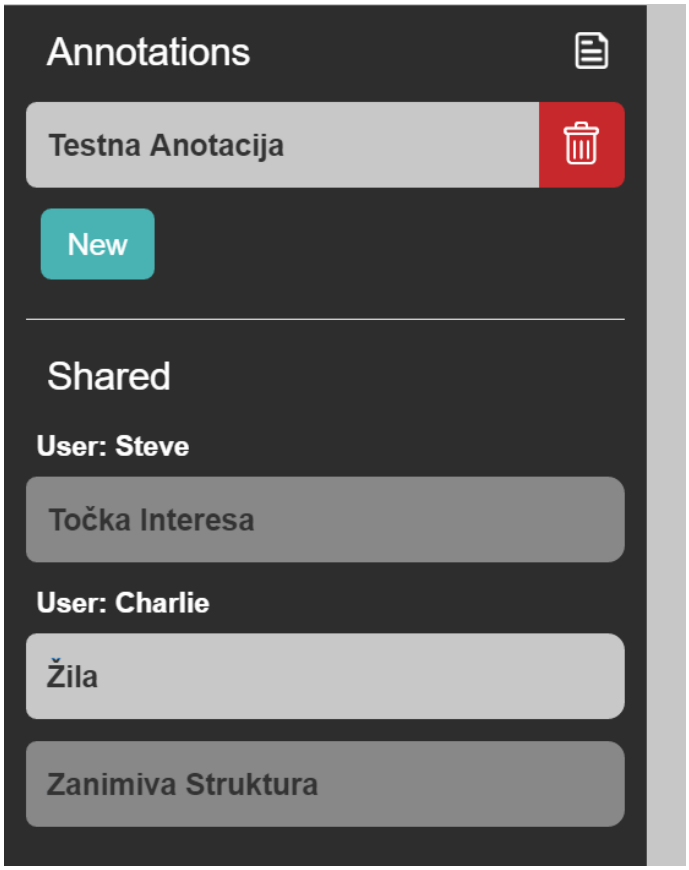
Slika 4.2: Slika prikazuje klepet med tremi uporabniki ogrodja, ki so udeleženi v isti seji.

4.3 Anotacije v vizualizaciji

Delitev scene in klepet pogosto ne zadostujeta potrebam uporabnikov vizualizacijskega ogrodja. Najpogosteje se namreč pojavi želja, da bi lahko ostalim uporabnikom izpostavili točke interesa na 3D vizualizaciji podatkov. Prav to nam omogočajo anotacije v vizualizaciji, katerih lokalno implementacijo smo že podrobneje predstavili v poglavju 3.3.5. Te smo v namen oddaljenega sodelovanja nekoliko prilagodili in razširili z delitvijo med vse uporabnike seje. Ker smo anotacije implementirali tako, da so vsi podatki centralizirani znotraj globalno dostopnega "singleton" objekta, jih lahko zelo enostavno sinhroniziramo s strežnikom in vsemi ostalimi udeleženci seje. Ob inicializaciji oddaljenega sodelovanja se poleg scene na strežnik prenesejo tudi vse anotacije gostujočega uporabnika, ki jih strežnik nato shrani med podatke seje. Nato se v fazi sinhronizacije začnejo sinhronizirati anotacije vseh uporabnikov. Te se tako na strežniku kot pri posameznemu uporabniku shranjujejo ločeno, tako da lahko vodimo dobro evidenco o njihovem lastništvu. Anotacije vsakega uporabnika se prikazujejo ločeno od anotacij drugih uporabnikov, hkrati pa so označene z uporabniškim imenom, kot je prikazano na sliki 4.3, kar omogoča lažji in učinkovitejši pregled nad zapiski različnih uporabnikov. Ker pa lahko zaradi velikega števila sodelujočih uporabnikov in posledično velikega števila anotacij vizualizacija postane nepregledna, smo v ogrodju omogočili lokalno skrivanje anotacij vseh uporabnikov. To nam omogoča sistematično pregledovanje zapiskov različnih uporabnikov, ne da bi nas pri tem ovirale anotacije ostalih uporabnikov.

4.4 Performančna analiza komunikacije

Pri perfromančni analizi komunikacije smo se osredotočili na analizo sinhronizacije kamere med uporabniki, udeleženimi v isti seji. Sinhronizacija je ključnega pomena, saj se mora izvajati realnočasovno v čim krajšem času (vsaj 20-krat na sekundo), tako da se kamera premika z minimalnim zaostankom in da opazovalci dobijo vtis, da se zvezno premika oziroma vrti v



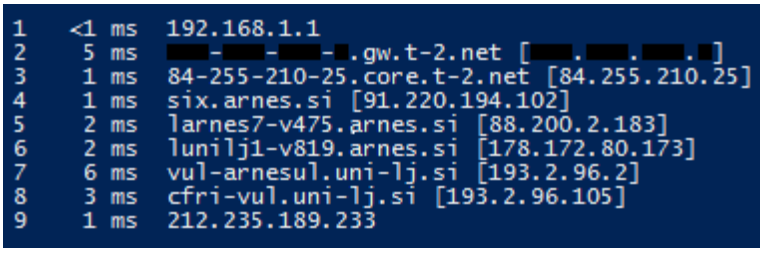
Slika 4.3: Slika prikazuje pikaz deljenih anotacij v stranski vrstici.

prostoru. Performančno analizo sihronizacije kamere smo opravili s pomočjo oddaljenega strežnika z naslednjimi specifikacijami:

- Procesor: Štirijederni procesor Intel Xeon L5520 (2266 MHz),
- Pomnilnik: 62 GB DDR3 (1333 MHz),
- Operacijski sistem: Ubuntu 16.04 LTS.

Ta je gostil strežniški del ogrodja s platformo Node.js in omogočal, da je preko njega potekalo oddaljeno sodelovanje. Uporabnike smo simulirali s pomočjo drugega računalnika, ki ga je s strežnikom ločevalo osem vmesnih vozlišč, kot je prikazano na sliki 4.4, ki prikazuje omrežno pot do strežnika in

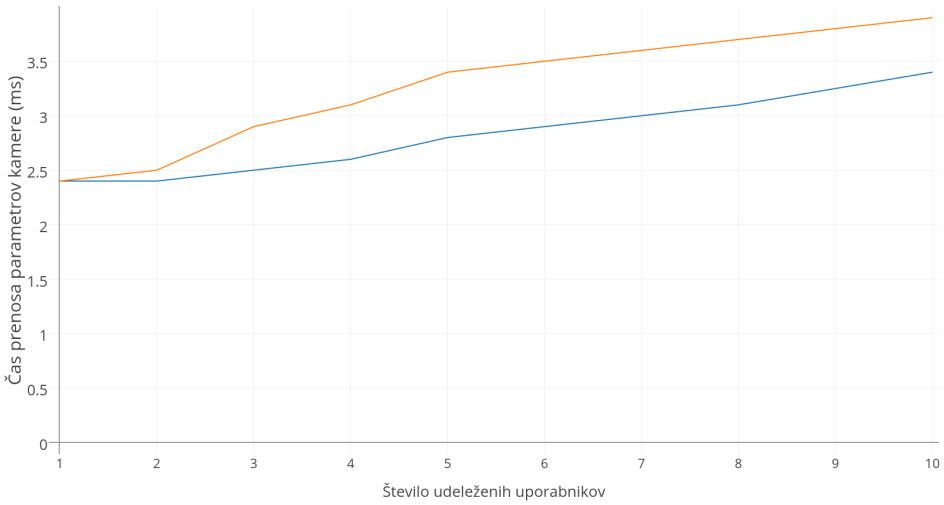
odzivne čase posameznih vozlišč na poti, zajete s pomočjo orodja Traceroute. Prav tako lahko razberemo, da ICMP paket za pot do strežnika in nazaj potrebuje približno 1 ms. Ta čas nam lahko služi kot referenčna točka pri evalvaciji sinhronizacije kamere med uporabniki.



Slika 4.4: Slika prikazuje časovne zamike pri prenosu ICMP paketa do vsakega izmed vozlišč na poti do ciljnega strežnika.

Tekom testiranja je en uporabnik gostil sejo, v to pa se je za različne teste povezalo 1, 2, 3, 4, 5, 8 in 10 uporabnikov in začelo spremljati kamero gostitelja. Ta je ob premikanju in vrtenju kamere pošiljal posodobitve parametrov kamere, ki jih je označil s časovno znamko. Posodobitve so se nato preko strežnika razposlale vsem uporabnikom, udeleženim v seji, ki so s pomočjo časovne znamke izračunali čas potovanja posodobitve od gostitelja do njih. Z opisanim postopkom smo za vsakega izmed udeleženih uporabnikov pridobili povprečen čas potovanja, tako da smo povprečili 1000 časov. Povprečje povprečnih časov in maksimalni povprečni čas smo za različno število uporabnikov predstavili z grafom na sliki 4.5.

Iz rezultatov je razvidno, da je maksimalni časovni zamik pri desetih sodelujočih uporabnikih samo 4 ms, kar pomeni, da lahko kamero osvežimo 250-krat na sekundo. To je več kot dovolj za tekoče premikanje in vrtenje z minimalnim zaostankom.



Slika 4.5: Slika prikazuje povprečen (modra črta) in maksimalen (oranžna črta) časovni zamik v relaciji s številom uporabnikov pri sinhroniziranju parametrov kamere z vsemi uporabniki, ki so udeleženi v seji in prejemajo parametre kamere.

Poglavje 5

Zaključek

V tem delu smo predstavili spletno vizualizacijsko ogodje Med3D, ki omogoča upodabljanje 3D mrežnih in volumetričnih medicinskih podatkov. Omogoča nam interakcijo z upodobljenimi podatki ter delno tudi njihovo obdelavo s pomočjo metod, uporabljenih pri posrednem upodabljanju. Ogrodje je razvito z namenom širše dostopnosti vizualizacije 3D medicinskih podatkov ter možnostjo oddaljenga sodelovanja med uporabniki, kar ga loči od ostalih predhodno razvitih ogrodij, ki sodelovanja ne omogočajo. V delu smo predstavili različne sestavne dele in funkcionalnosti ogrodja, ki obsegajo različne pristope upodabljanja, odloženo upodabljanje, kompleksen uporabniški vmesnik, ki uporabniku omogoča učinkovito upravljanje ogrodja, ter oddaljeno sodelovanje. To obsega delitev podatkov, pogleda, anotacij in sporočil z ostalimi uporabniki ogrodja.

Poleg ogrodja smo predstavili tudi performančno analizo upodabljanja, s pomočjo katere si lahko lažje zastavimo vse nadaljnje cilje, ki jih je mogoče realizirati v okviru zmožnosti spletnega upodabljanja. Poudarek smo dali tudi na posredno upodabljanje s pomočjo algoritma Marching cubes, ki smo ga podrobno analizirali in predstavili ter primerjali štiri različne implementacije, od tega dve spletni implementaciji (Javascript, ASM.js) in dve domorodni implementaciji (C++, Java). Poleg lokalnih funkcionalnosti ogrodja smo v analizo vključili tudi oddaljeno sodelovanje, pri katerem smo izmerili

hitrost sinhronizacije pogleda med uporabniki, kar je ključnega pomena za dobro uporabniško izkušnjo.

5.1 Nadaljnje delo

Z zaključkom tega dela pa se ne bo ustavil tudi nadaljnji razvoj ogrodja. Imamo še številne cilje in nove ideje, s katerimi bomo v nadaljnjem razvoju izpopolnili ogrodje. To želimo razširiti z različnimi pristopi neposrednega upodabljanja, kot na primer Ray casting in Splatting, ki smo ju omenili v poglavju 3.2.2. Ker se ogrodje izvaja znotraj spletnega brskalnika, imamo dostop zgolj do omejenih virov. Zato želimo v prihodnosti v primeru zahtevnejših vizualizacij in slabših uporabniških naprav (mobilne naprave) omogočiti izvajanje zahtevnejših izračunov na strežniku, kar lahko zajema obdelavo in pretvorbo podatkov ali celostno oddaljeno upodabljanje. To nam hkrati odpre možnost razvoja kompleksnejših metod za posredno in neposredno upodabljanje volumetričnih podatkov. Ena izmed metod, ki jih želimo razviti, je MPUI. To bomo tako kot Marching cubes implementerali s pomočjo programskih jezikov Javascript, njegove podmnožice ASM.js in C++ ter vse implementacije učinkovito paralelizirali, saj algoritem zaradi uporabe lokalnih aproksimacij omogoča visoko stopnjo vzporednosti. Hkrati želimo izvajanje dodatno pohitriti s pomočjo ogrodja OpenCL [49], ki omogoča hitrejše izvajanje operacij, ki jih je mogoče dobro paralelizirati z uporabo grafičnega pospeševalnika. Pri implementaciji si bomo pomagali z diplomskim delom [21] Miloša Lukića, v katerem je analiziral algoritem MPUI in ovrednotil možnosti paralelizacije.

Literatura

[1] V. R. Kamat and J. C. Martinez, "Visualizing simulated construction operations in 3D," *Journal of computing in civil engineering*, vol. 15, no. 4, pp. 329–337, 2001.

[2] T. V. Papathomas, J. A. Schiavone, and B. Julesz, "Applications of computer graphics to the visualization of meteorological data," in *ACM SIGGRAPH Computer Graphics*, vol. 22, no. 4. ACM, 1988, pp. 327–334.

[3] B. Wieczorek, A. Sobieraj, and K. Pajak, "Modelling and computer animation of geodetic field work," *14th SGEM GeoConference on Informatics, Geoinformatics and Remote Sensing*, vol. 1, no. SGEM2014 Conference Proceedings, ISBN 978-619-7105-10-0/ISSN 1314-2704, June 19-25, 2014, Vol. 1, pp. 691–698, 2014.

[4] A. Hassan and C. J. Fluke, "Scientific visualization in astronomy: Towards the petascale astronomy era," *Publications of the Astronomical Society of Australia*, vol. 28, no. 02, pp. 150–170, 2011.

[5] L. Chittaro, "Information visualization and its application to medicine," *Artificial intelligence in medicine*, vol. 22, no. 2, pp. 81–88, 2001.

[6] B. Preim and D. Bartz, *Visualization in medicine: theory, algorithms, and applications*. Morgan Kaufmann, 2007.

[7] C. Bohak, S. Žagar, A. Sodja, P. Škrlj, U. Mitrović, F. Pernuš, and M. Marolt, "Neck veins: an interactive 3D visualization of head veins,"

in *Proceedings of the 4th International Conference World Usability Day Slovenia 2013, 25 Nov, Ljubljana, Slovenia, E. Stojmenova (Ed.)*, 2013, pp. 64–66.

[8] L. Bagrow, *History of cartography.* Transaction Publishers, 2010.

[9] M. Friendly and D. J. Denis, "Milestones in the history of thematic cartography, statistical graphics, and data visualization," *U RL http://www. datavis. ca/milestones*, 2001.

[10] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," *SIGGRAPH Comput. Graph.*, vol. 22, no. 4, pp. 65–74, Jun. 1988.

[11] C. R. Crawford and K. F. King, "Computed tomography scanning with simultaneous patient translation," *Medical Physics*, no. 17, pp. 967 – 982, 1990.

[12] W. A. Kalender, W. Seissler, E. Klotz, and P. Vock, "Spiral volumetric CT with single-breath-hold technique, continuous transport and continuous scanner rotation," *Radiology*, no. 176, pp. 181 – 183, 1990.

[13] J. M. Ollinger and J. A. Fessler, "Positron-emission tomography," *IEEE Signal Processing Magazine*, vol. 14, no. 1, pp. 43–55, 1997.

[14] P. A. Rinck, *Magnetic Resonance in Medicine. The Basic Textbook of the European Magnetic Resonance Forum. 9th edition.* TRTF, 2016, vol. 9.1, e-Version.

[15] D. Krakow, J. Williams, M. Poehl, D. L. Rimoin, and L. D. Platt, "Use of three-dimensional ultrasound imaging in the diagnosis of prenatal-onset skeletal dysplasias," *Ultrasound in Obstetrics and Gynecology*, vol. 21, no. 5, pp. 467–472, 2003. [Online]. Available: http://dx.doi.org/10.1002/uog.111

[16] "SimVascular," http://simvascular.github.io/, (Accessed on 08/15/2016).

[17] T. Kroes, F. H. Post, and C. P. Botha, "Exposure render: An interactive photo-realistic volume rendering framework," *PLoS ONE*, vol. 7, no. 7, pp. 1–10, Jul. 2012.

[18] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, Aug. 1987.

[19] J. Bloomenthal, "Graphics gems IV," P. S. Heckbert, Ed. San Diego, CA, USA: Academic Press Professional, Inc., 1994, ch. An Implicit Surface Polygonizer, pp. 324–349. [Online]. Available: http://dl.acm.org/citation.cfm?id=180895.180923

[20] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel, "Multi-level partition of unity implicits," in *ACM SIGGRAPH 2005 Courses*, ser. SIGGRAPH '05. New York, NY, USA: ACM, 2005. [Online]. Available: http://doi.acm.org/10.1145/1198555.1198649

[21] M. Lukić, "Primerjava algoritmov za rekonstrukcijo 3D modelov iz volumetričnih podatkov," 2014.

[22] M. Meißner, H. Pfister, R. Westermann, and C. Wittenbrink, "Volume visualization and volume rendering techniques," *Eurographics tutorial*, 2000.

[23] M. Ament, "Volume rendering."

[24] "Çelebi tutorial — volume rendering," http://www.byclb.com/TR/Tutorials/volume_rendering/Index.aspx, (Accessed on 08/24/2016).

[25] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," in *Proceedings of the 1994 symposium on Volume visualization.* ACM, 1994, pp. 91–98.

[26] O. Wilson, A. VanGelder, and J. Wilhelms, "Direct volume rendering via 3D textures," Santa Cruz, CA, USA, Tech. Rep., 1994.

[27] L. Westover, "Footprint evaluation for volume rendering," in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '90. New York, NY, USA: ACM, 1990, pp. 367–376. [Online]. Available: http://doi.acm.org/10.1145/97879.97919

[28] L. A. Westover and L. A. Westover, "Splatting : A parallel , feed-forward volume rendering algorithm splatting : A parallel , feed-forward volume rendering algorithm by," 1991.

[29] C. Zhang and R. Crawfis, "Volumetric shadows using splatting," in *Visualization, 2002. VIS 2002. IEEE.* IEEE, 2002, pp. 85–92.

[30] P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '94. New York, NY, USA: ACM, 1994, pp. 451–458. [Online]. Available: http://doi.acm.org/10.1145/192161.192283

[31] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

[32] "WebGL," https://www.khronos.org/webgl/, (Accessed on 08/14/2016).

[33] "OpenGL shading language," https://www.opengl.org/documentation/glsl/, (Accessed on 08/14/2016).

[34] "Three.js - Javascript 3D library," http://threejs.org/, (Accessed on 08/15/2016).

[35] "BabylonJS," http://www.babylonjs.com/, (Accessed on 08/15/2016).

[36] B. Green and S. Seshadri, *AngularJS.* "O'Reilly Media, Inc.", 2013.

[37] "Bootstrap · the world's most popular mobile-first and responsive front-end framework." http://getbootstrap.com/, (Accessed on 08/14/2016).

[38] "Cascading style sheets," https://www.w3.org/Style/CSS/, (Accessed on 08/17/2016).

[39] "Node.js," https://nodejs.org/en/, (Accessed on 08/14/2016).

[40] "ParaView," http://www.paraview.org/overview/, (Accessed on 08/15/2016).

[41] "ParaViewWeb," http://paraviewweb.kitware.com/, (Accessed on 08/15/2016).

[42] "MongoDB," https://www.mongodb.com/, (Accessed on 08/15/2016).

[43] "ASM.js," http://asmjs.org/, (Accessed on 08/17/2016).

[44] "Web Workers," https://www.w3.org/TR/workers/, (Accessed on 08/17/2016).

[45] "Cheerp - C++ for the Web," http://leaningtech.com/cheerp/, (Accessed on 08/17/2016).

[46] "Emscripten," https://github.com/kripken/emscripten/wiki, (Accessed on 08/17/2016).

[47] "HTML," https://www.w3.org/TR/html/, (Accessed on 08/17/2016).

[48] "Scalable vector graphics (SVG) 1.1 (second edition)," https://www.w3.org/TR/SVG/, (Accessed on 08/23/2016).

[49] "OpenCL - The open standard for parallel programming of heterogeneous systems," https://www.khronos.org/opencl/, (Accessed on 08/17/2016).